



Framework eMobc Android

Manual del Usuario

Versión 0.1

AVISO LEGAL

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a. **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(g) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

d. **"License Elements"** means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.

e. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

f. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

g. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a

compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

h. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

i. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

j. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. **Fair Dealing Rights.** Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. **License Grant.** Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

d. to Distribute and Publicly Perform Adaptations.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights described in Section 4(e).

4. **Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as

required by Section 4(d), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(d), as requested.

b. You may Distribute or Publicly Perform an Adaptation only under: (i) the terms of this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-NonCommercial-ShareAlike 3.0 US) ("Applicable License"). You must include a copy of, or the URI, for Applicable License with every copy of each Adaptation You Distribute or Publicly Perform. You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License. You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

d. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, (iv) consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(d) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

e. For the avoidance of doubt:

i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers

voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c).

f.Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING AND TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THIS EXCLUSION MAY NOT APPLY TO YOU.

6.Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a.This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b.Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
- b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
- c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
- f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Índice de contenidos

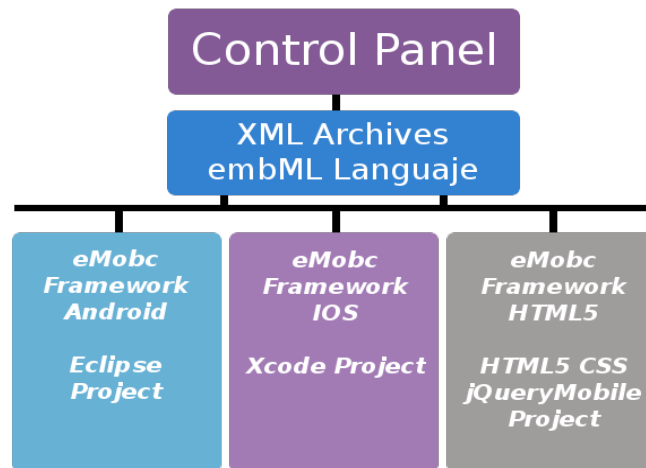
<u>Pág.</u>		<u>Sección</u>
2	Aviso Legal y Licencia del Documento	Licencias
8	Índice de Contenidos	Índice de Contenidos
10	Descripción General	Descripción General
11	• Archivo App.xml	Descripción General
11	• Levels y NextLevels	Descripción General
12	• Funcionamiento de la Aplicación	Descripción General
13	• Level	Descripción General
	• Estructura de las aplicaciones	Descripción General
16	• Asociación de datos de los Level	Descripción General
	• ApplicationData y AppLevelData	Descripción General
19	Descripcion de los parsers de Android	Parsers
	• ParseUtils	
39	<u>Tipos de Actividades (Pantallas)</u>	Actividades
	• Splash	Actividades
	• Cover	Actividades
	• Galería de Imágenes	Actividades
	• Lector PDF	Actividades
	• Lector QR	Actividades
	• Contenedor Web	Actividades
	• Lista	Actividades
	• Imagen con Lista	Actividades
	• Video	Actividades
	• Mapa	Actividades
	• Búsqueda	Actividades
	• Formulario	Actividades
	• Texto con Imagen	Actividades
	• Imagen con Zoom	Actividades
	• Multimedia	Actividades
	• Calendario	Actividades
85	Como crear una nueva pantalla en Android	Crear Nueva Pantalla
	• Pasos para crear una nueva pantalla	Crear Nueva Pantalla
	• Esquema de como crear una nueva pantalla	Crear Nueva Pantalla
96	Actividad de Terceros	Actividad de Terceros

página en blanco – separador

Descripción general

eMobic es un framework que permite generar aplicaciones móviles definidas desde un panel de control para distintas plataformas: Android, iOS y HTML5.

El framework está basado en archivos xml. Una vez se ha definido la aplicación que se quiere crear desde el panel de control, éste genera una serie de archivos xml que describen todas las propiedades, ventanas, y funciones que tendrá la aplicación.



Cada plataforma dispone de un proyecto base donde se almacenarán estos archivos xml. Durante la ejecución de la aplicación, son parseados a medida que son necesarios para ir generando la aplicación a partir de la información recopilada.

El código necesario para ejecutar la aplicación ya está disponible en los proyectos base. Tan solo es necesario añadir los archivos xml y compilar el proyecto para obtener una aplicación funcional.

App.xml

El archivo xml principal es el app.xml (Ver application data). Toda la información general que se aplica a la aplicación está en este archivo como el tipo de publicidad, las rotaciones soportadas, los menús de navegación (ver menús) o el Entry Point (ver Entry Point).

El EntryPoint básicamente es la pantalla que se mostrará en primer lugar (por defecto será la portada).

Levels y NextLevels

Los XML definen tanto la aplicación en general (app.xml) como cada una de sus pantallas. Estas pantallas están estructuradas en el framework como levels. Un level es una ventana de la aplicación. Es de un tipo concreto y tiene unos datos asociados.

Todos los levels que contiene una aplicación estarán definidos en su app.xml.

Las acciones dentro de las pantallas de la aplicación, como pulsar un botón o una imagen, pueden tener asociados un Next Level. Un Next Level es la próxima ventana que se mostrará cuando el usuario realice la acción a la que está asociado. (ver Levels)

```
<levels>
  <!-- web -->
  <level>
    <levelId>web</levelId>
    <levelTitle>Ventana Web</levelTitle>
    <levelFile>web.xml</levelFile>
    <levelType>WEB_ACTIVITY</levelType>
  </level>
```

Funcionamiento de la aplicación

El funcionamiento de una aplicación construida con el framework se basa principalmente en parseado de los archivos XML que definen la aplicación. El primer archivo que se parsea es app.xml. Se generarán todos los datos de la aplicación.

Una vez la portada o el entry point sea visible, cada acción del usuario puede llevar a nuevas ventanas (Next Level) o a salir de la aplicación.

Cuando se realiza una acción que tiene asociado un **Next Level**, la aplicación busca el archivo de datos xml asociado a ese **Next Level** y lo parsea para recuperar la información y crear la pantalla que se va a mostrar. (Ver Levels).

Conviene diferenciar, tanto en Android como en iOS, la creación de la aplicación desde el panel de control del funcionamiento. El panel de control es el que genera los archivos xml y el proyecto en Android e iOS es el que va construyendo, a partir de esos xml, la aplicación en tiempo de ejecución.

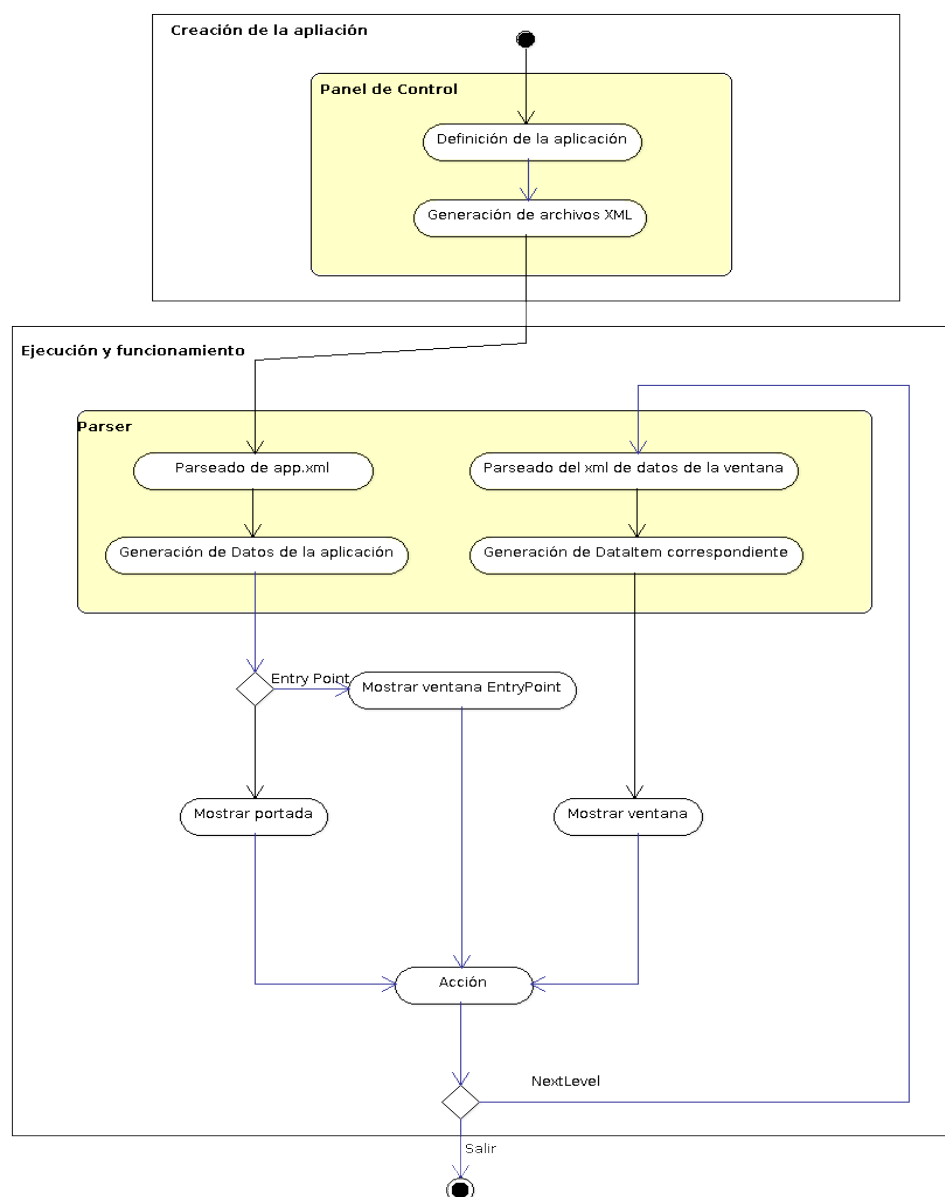


Figura 1: Secuencia de Ejecución de una aplicación eMobic.

Level

Estructura de las aplicaciones

Las aplicaciones que genera el framework se estructuran en base a **Levels** (niveles). Un level es un tipo de ventana que tiene la aplicación. Por esa razón se incluyen todos los levels de una aplicación en el app.xml.

Next Level

Cuando en una ventana tiene un botón, y al pulsarlo tiene asociado un enlace a una nueva ventana, decimos que tiene un **Next Level** asociado a la acción de pulsar el botón.

Un Next Level es el level que se creará cuando se realiza una acción. Para definir un Next Level es necesario tanto el level que queremos crear, como los datos asociados a él que la aplicación tendrá que cargar.

Portada.xml

```
<button>
  <buttonTitle>Navegador</buttonTitle>
  <buttonFileName>images/buttonw.png</buttonFileName>
  <nextLevel>
    <nextLevelLevelId>web</nextLevelLevelId>
    <nextLevelDataId>web1</nextLevelDataId>
  </nextLevel>
</button>
```

Diferencia entre level y ventana

Un level no es una de todas las pantallas que tendrá la aplicación. Si una aplicación tiene tres tipos de ventana diferentes (por ejemplo: calendario, lista con imágenes y lector pdf) tendrá tres levels definidos en su app.xml. Sin embargo, otra aplicación puede tener en total 3 ventanas, todas ellas de tipo contenedor web y tan solo tendrá un level declarado en su app.xml.

Para diferenciar cada una de estas 3 ventanas entre sí, porque previsiblemente queremos que contengan diferente información, se utilizan los datos asociados al level, definidos mediante el tag “nextLevelDataId” de la llamada al Next Level (ver figura 1).

```
<levels>
  <!-- web -->
  <level>
    <levelId>web</levelId>
    <levelTitle>Ventana Web</levelTitle>
    <levelFile>web.xml</levelFile>
    <levelType>WEB_ACTIVITY</levelType>
  </level>
```

Asociación de datos de los levels

Para identificar cada level y sus datos se utilizan identificadores de tipo String. Un level tiene un identificador único y un archivo xml de datos asociado. Cada vez que se quiere crear un Next Level se utiliza el identificador del level (que está definido en el app.xml) y el identificador de los datos.

Cada vez que se llama a un Next Level desde la aplicación, se consulta la información del app.xml por levelId y se leen los datos identificados por el tag “nextLevelDataId” dentro del archivo xml especificado en el tag “levelFile”.

Un archivo de datos xml puede tener varios datos dentro. Cada uno tiene un identificador único al que se puede hacer referencia en cada llamada a un Next Level.

A continuación se muestra un ejemplo para clarificar el funcionamiento de los levels y los datos asociados. Continuando con el escenario anterior, en el que se tienen tres ventanas de tipo contenedor web, vamos a ver cómo se referencian entre sí los tags y los archivos xml.

Primero se presentarán cómo son nuestros archivos xml y qué significa cada tag dentro de ellos.

Este es un fragmento de Portada.xml. Es la definición de una ventana de portada que tiene, entre otras cosas, un botón de título Navegador con un Next Level asociado a una ventana de contenedor web.

La etiqueta <nextLevelLevelId> tiene el identificador del level al que se irá si se pulsa el botón

La etiqueta <nextLevelDataId> tiene el identificador de los datos que el level debe cargar cuando se cree.

```
Portada.xml
...
<button>
  <buttonTitle>Navegador</buttonTitle>
  <buttonFileName>images/buttonw.png</buttonFileName>
  <nextLevel>
    <nextLevelLevelId>web</nextLevelLevelId>
    <nextLevelDataId>web1</nextLevelDataId>
  </nextLevel>
</button>
...
```

Este es un fragmento del archivo app.xml donde se declaran los levels de nuestra aplicación.

Como solo tendrá ventanas de tipo web, solo aparece un level. La etiqueta <levelId> contiene el identificador del level.

La etiqueta <levelFile> contiene el archivo xml que almacena los datos del level.

La etiqueta <levelType> contiene el tipo de ventana que corresponde al level (en este caso web).

```
App.xml
...
<levels>
  <!-- web -->
  <level>
    <levelId>web</levelId>
    <levelTitle>Ventana Web</levelTitle>
    <levelFile>web.xml</levelFile>
    <levelType>WEB_ACTIVITY</levelType>
  </level>
</levels>
...
```

Este es un fragmento del archivo de datos del level web. Por simplicidad se ha reducido el número de ventanas web.

La etiqueta <dataId> contiene el identificador de los datos dentro del archivo xml de datos.

La etiqueta <local> establece si hay que buscar una dirección en la web o en un archivo local.

La etiqueta <webUrl> contiene la dirección web o el archivo html local que cargará el contenedor web.

(Para más información de los tags de los archivos xml consulta Archivos xml).

Una vez explicados todos archivos xml que van a intervenir en el ejemplo, se mostrarán las interacciones entre ellos.

```
Web.xml
...
<levelData>
  <data>
    <dataId>web1</dataId>
    <headerImageFile>images/web.png</headerImageFile>
    <headerText>Ventana Web</headerText>
    <local>true</local>
    <webUrl>archivo.html</webUrl>
  </data>
  <data>
    <dataId>web2</dataId>
    <headerImageFile>images/web.png</headerImageFile>
    <headerText>Ventana Web</headerText>
    <local>false</local>
    <webUrl>www.google.es</webUrl>
  </data>
</levelData>
...
```

Actualmente estamos en el escenario en el que el usuario pulsa un botón de la portada que tiene asociado un Next Level a una ventana web que cargará un archivo html local (archivo.html). En ese momento se busca el level en el app.xml por la etiqueta <nextLevelLevelId>.

Portada.xml

```
<button>
  <buttonTitle>Navegador</buttonTitle>
  <buttonFileName>images/buttonw.png</buttonFileName>
  <nextLevel>
    <nextLevelLevelId>web</nextLevelLevelId>
    <nextLevelDataId>web1</nextLevelDataId>
  </nextLevel>
</button>
```

App.xml

```
<levels>
  <!-- web -->
  <level>
    <levelId>web</levelId>
    <levelTitle>Ventana Web</levelTitle>
    <levelFile>web.xml</levelFile>
    <levelType>WEB_ACTIVITY</levelType>
  </level>
</levels>
```

El level al que se ha hecho referencia dentro del app.xml, tiene asociado un archivo xml de datos en el tag <levelFile>. Se busca ese archivo y dentro de él aquellos datos a los que se hace referencia con el tag <nextLevelDataId> del archivo portada.xml.

portada.xml

```
<button>
  <buttonTitle>Navegador</buttonTitle>
  <buttonFileName>images/buttonw.png</buttonFileName>
  <nextLevel>
    <nextLevelLevelId>web</nextLevelLevelId>
    <nextLevelDataId>web1</nextLevelDataId>
  </nextLevel>
</button>
```

app.xml

```
<levels>
  <!-- web -->
  <level>
    <levelId>web</levelId>
    <levelTitle>Ventana Web</levelTitle>
    <levelFile>web.xml</levelFile>
    <levelType>WEB_ACTIVITY</levelType>
  </level>
</levels>
```

web.xml

```
<levelData>
  <data>
    <dataId>web1</dataId>
    <headerImageFile>images/web.png</headerImageFile>
    <headerText>Ventana Web</headerText>
    <local>true</local>
    <webUrl>archivo.html</webUrl>
  </data>
  <data>
    <dataId>web2</dataId>
    <headerImageFile>images/web.png</headerImageFile>
    <headerText>Ventana Web</headerText>
    <local>false</local>
    <webUrl>www.google.es</webUrl>
  </data>
</levelData>
```

La segunda ventana cargará la dirección web `http://www.google.es` dentro del contenedor web. Hay que incluir en el archivo que define la portada (`portada.xml`) otro botón para que cargue los datos de la nueva ventana.

De esta manera, cada vez que se pulse el segundo botón, se accederá al mismo archivo de datos xml (`web.xml`) pero al nuevo identificar de datos que ahora contendrá la información para que se cargue google.

```
portada.xml
<button>
  <buttonTitle>Navegador</buttonTitle>
  <buttonFileName>images/buttonw.png</buttonFileName>
  <nextLevel>
    <nextLevelLevelId>web</nextLevelLevelId>
    <nextLevelDataId>web1</nextLevelDataId>
  </nextLevel>
</button>
<button>
  <buttonTitle>Navegador</buttonTitle>
  <buttonFileName>images/buttonw.png</buttonFileName>
  <nextLevel>
    <nextLevelLevelId>web</nextLevelLevelId>
    <nextLevelDataId>web2</nextLevelDataId>
  </nextLevel>
</button>
```

AplicationData y AppLevelData.

Hasta ahora hemos visto que para consultar levels y sus archivos de datos xml, se accede al `app.xml`. Esto no es del todo cierto. Cuando se necesita acceder a los datos contenidos en un archivo xml, primero se parsea el archivo y se transforma en un objeto que al que luego se puede acceder mediante código.

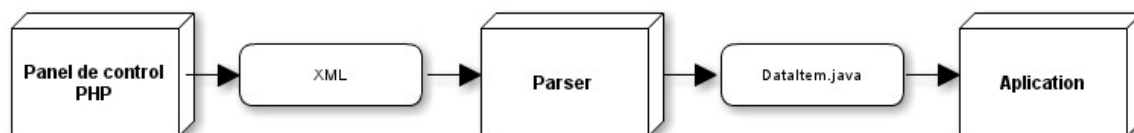


Figura 2: Flujo de generación de una aplicación eMobic.

Como puede verse en la figura 2, los XML los genera el panel de control pero la aplicación en sí no accede a ellos. Necesita un objeto que almacene esa información y al que se pueda acceder desde código. Aunque realmente el parser es parte de la aplicación, trabaja como una entidad independiente y cuando es necesario genera estos objetos que la aplicación utilizará para construir las ventanas. (Ver parser).

Aquí es donde entran en juego los **AplicationData** y los **AppLevelData**.

AplicationData : es el objeto que almacenará la información del `app.xml`. Previsiblemente habrá solo uno, aunque es posible generarlo manera dinámica en algunas ventanas como en los formularios.

AppLevelData : es el objeto que almacena la información de cada uno de los archivos xml. A su vez y como cada archivo de datos xml puede tener un conjunto de datos demasiado complejos para almacenarse en un tipo simple, es posible que sea necesario utilizar estructuras de datos complejas. Para estos nuevos tipos de datos se utilizan los **DataItem**.

La importancia de la clase ApplicationData

La clase Application data es la clase principal, o dicho de otra manera, el módulo central donde están definidas todas las características de la aplicación, provenientes del fichero app.xml. Es en ésta clase donde estan definidos:

- Todos los niveles (levels) de la aplicación
- El Entry Point
- El tipo de rotación activa
- Los distintos tipos de menú
- Los temas y estilos

Gracias a esta clase, se puede acceder desde cualquier otra, a información genérica de la aplicación.

Por ejemplo, en la generación de un tipo de pantalla en tiempo de ejecución, es necesario conocer información básica de la aplicación, como por ejemplo si existen o no menús. De esa manera, en su propio generador, se tomará una decisión u otra dependiendo de las características que estén precargadas en la clase ApplicationData.

Es por ello, por lo que la clase ApplicationData debe estar accesible desde cualquier punto. Por tanto, en la aplicación existe un Singleton que hace referencia a ésta clase.

Dicho Singleton se encuentra en la pantalla SplashActivity, debido a que se ejecuta siempre la primera al comenzar la aplicación.

Cuando el programa se inicia, automáticamente se genera un objeto de la clase ApplicationData por defecto. Acto seguido, se hace un parseo del fichero app.xml, y se genera toda la información almacenada en éste, de manera que se tiene un objeto de la clase, con todos los datos necesarios.

Como plus, se pueden almacenar características de otros ficheros xml, como el de temas y estilos, que también cuentan con información que pueden requerir otras clases.

De esta manera, existe una única instancia en la aplicación, donde está la información de diversos xml, por lo que solamente es necesario un único parseo de los ficheros.

El siguiente código muestra un segmento de la clase SpashActivity, donde se parsean el app.xml y otros xml útiles.

```
try {  
    INSTANCE = ApplicationData.readApplicationData(this);  
    INSTANCE.setLevelStyleTypeMap(ParseUtils.parseStylesData(this,  
    INSTANCE.getStylesFileName()));  
    INSTANCE.setFormatStyleMap(ParseUtils.parseFormatData(this,  
    INSTANCE.getFormatsFileName()));  
} catch (InvalidFileException e) {  
    Log.e("SplashActivity", e.toString());  
}
```

Y a parte, existe el metodo getApplicationData(), que devuelve la instancia al objeto:

```
public static ApplicationData getApplicationData() {  
    return INSTANCE;  
}
```

Por lo que para acceder al contenido de ApplicationData desde cualquier clase simplemente se accederá de la siguiente manera:

```
ApplicationData applicationData = SplashActivity.getApplicationData();
```

Veamos un ejemplo práctico:

Si nos fijamos en cualquier clase de Activity, como ImageTextDescriptionActivity, en su método onCreate(), se puede

encontrar la siguiente secuencia:

```
ApplicationData applicationData = SplashActivity.getApplicationData();
if(applicationData != null){
    Intent intent = getIntent();
    isEntryPoint = (Boolean)intent.getSerializableExtra(
        ApplicationData.IS_ENTRY_POINT_TAG);
    NextLevel Next Level = (NextLevel)intent.getSerializableExtra(
        ApplicationData.NEXT_LEVEL_TAG);
    ActivityGenerator generator = applicationData.getFromNextLevel(
        this, Next Level);
    generator.initializeActivity(this);
}
```

En el método se accede a la instancia del objeto ApplicationData de la pantalla SplashActivity, y se toma un generator a partir de dicha clase, que no es más que la traducción que se ha hecho de los ficheros XML.

Descripción del Parser

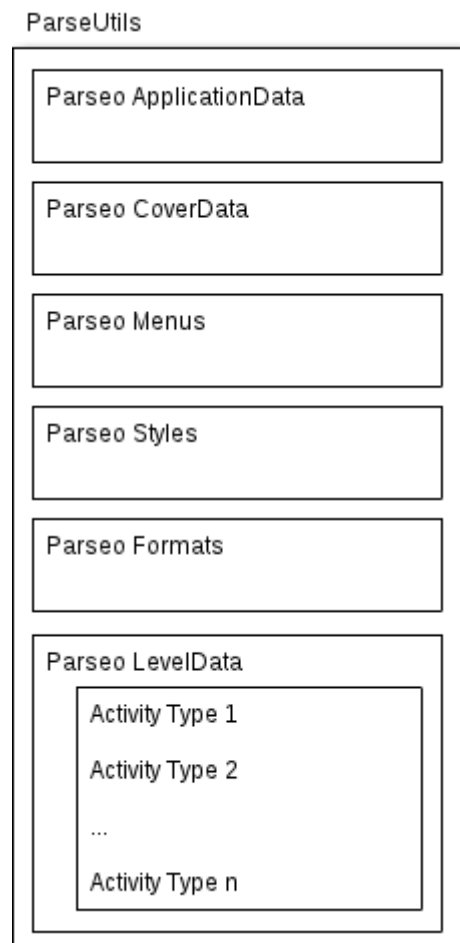
Explicación y desarrollo

La aplicación cuenta con un sistema de parseo, debido a la importancia que se le da a dinamización de las activities. Debido a que las pantallas de la aplicación no están predeterminadas de forma estática, y se van generando dependiendo de lo leído en los ficheros XML, es necesario un sistema que interprete sus contenidos, y los traduzca a objetos interpretables en java.

Es por ello por lo que existe una clase llamada ParseUtils. En ésta clase estan implementados todos los metodos necesarios para parsear un fichero XML. Cualquier fichero XML que se encuentre en la carpeta assets (proveniente del panel de control), tendra su correspondiente lugar en la clase ParseUtils.

Para conocer en profundidad su funcionamiento hay que tener en cuenta dos conceptos:

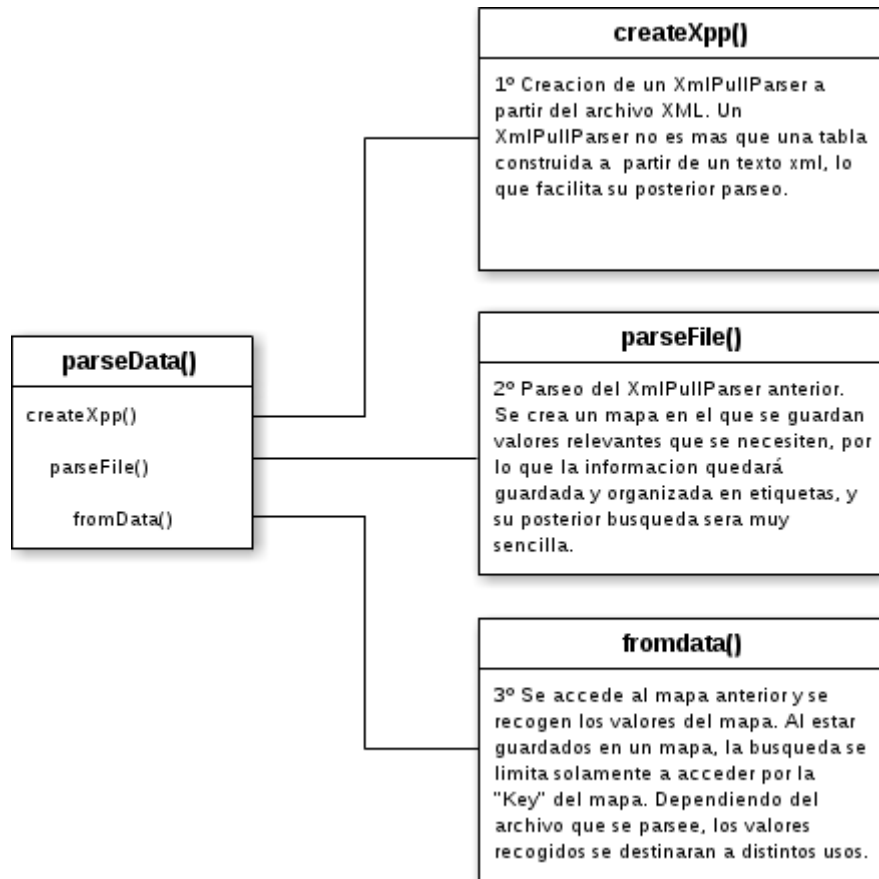
- La clase se divide por grupos (se explica mas tarde), en los que cada grupo representa el parseo de un tipo de xml.



- Cada uno de los grupos se divide en tres partes (o pasos):
 - Llamada al método createXpp(). Es el primer paso que se debe seguir para parsear un fichero xml. Se crea un XmlPullParser, que facilitara el parseo de los elementos
 - Llamada al metodo parse[Type]File(). Segundo paso a la hora de parsear. Utiliza el XmlPullParser

para recorrerlo, y generar un mapa en el que la clave es un nombre identificador, y el valor es lo que se ha leído del XmlPullParser. Por tanto, se irán guardando los datos relevantes para el tipo de pantalla.

- Llamada al metodo from[Type]Data(). Ultimo paso en el parseo del fichero. Una vez se tiene el mapa con los datos guardados, se retoman y se les proporciona una utilidad.



Una vez conocida su estructura general, se puede apreciar que para el parseo simplemente se necesitara llamar al metodo parse[Type]Data(), siendo [Type] el tipo de archivo que se quiere parsear. Este metodo necesita dos entradas imprescindibles:

- Context: que no es mas que el contexto de la actividad
- xmlFileName: que es una cadena de texto del nombre del fichero xml a parsear (nombre_del_fichero.xml)

El cometido principal de esta clase, como ya se ha dicho, es recoger informacion de ficheros XML y transformarla a sintaxis legible para java/Android, imaginemos que contamos con el siguiente fichero XML:

ejemplo.xml

```

<application>
  <tag1>contenido_tag1</tag1>
  <tag2>
    <tag2_1>contenido_tag2_1</tag2_1>
    <tag2_2>contenido_tag2_2</tag2_2>
  </tag2>
</application>
    
```

Se puede apreciar un bloque `<application>` que contiene dos etiquetas `<tag1>` y `<tag2>`. A su vez, `<tag2>` contiene otros dos campos `<tag2_1>` y `<tag2_2>`.

Si queremos crear su parser, primeramente se debe crear en la clase `ParseUtils` el metodo `parseEjemplo1Data()`, que es el que se encargará de invocar a los tres metodos `createXpp()`, `parseFile()`, y `fromData()`.

El método `parseEjemplo1Data()` tendra como entrada, el contexto de la activity, y el nombre del fichero a parsear, en este caso "ejemplo.xml".

Como primero de los pasos, se crea un objeto `XmlPullParser` con el metodo `createXpp()`:

```
XmlPullParser xpp = createXpp(context, Locale.getDefault(), xmlFileName);
```

Una vez se tiene el `XmlPullParser`, habra que implementar el método que se invoca en el segundo paso, hablamos de `parse[Type]File()`, que se llamará en este caso `parseEjemplo1File()`.

```
Map<String, Object> data = parseEjemplo1File(xpp);
```

Este nuevo metodo tendra como entrada el `XmlPullParser` anteriormente creado.

Para implementarlo correctamente hay que fijarse en la estructura del fichero XML que vamos a parsear.

Para el ejemplo tendremos que reconocer 5 etiquetas: `<application>` `<tag1>` `<tag2>` `<tag2_1>` y `<tag2_2>`. Por lo que crearemos 5 constantes en la clase:

```
private static final String _APPLICATION_TAG_ = "application";
private static final String _TAG1_TAG_ = "tag_1";
private static final String _TAG2_TAG_ = "tag_2";
private static final String _TAG21TAG_ = "tag2_1";
private static final String _TAG22_TAG_ = "tag2_2";
```

Esas constantes deberan tomar el valor identico al que se presenta en el XML, ya que sera la forma de reconocer dicha cadena.

Una vez se tiene la base, se procede a implementar el metodo `parseEjemplo1File()`.

```

private static Map<String, Object> parseEjemplo1File(XmlPullParser xpp) {
    final Map<String, Object> ret = new HashMap<String, Object>();
    NwXmlStandarParser parser = new NwXmlStandarParser(xpp,
        new NwXmlStandarParserTextHandler() {
            private Object currItem;

            @Override
            public void handleText(String currentField, String text) {
                if(currentField.equals(_TAG1_TAG_)){
                    ret.put(currentField, text);
                }else if(currentField.equals(_TAG2_TAG_)){
                    currItem = new Object();
                    ret.put(currentField, currItem);
                }else if(currentField.equals(_TAG21_TAG_)){
                    currItem.setTag1(text);
                }else if(currentField.equals(_TAG22_TAG_)){
                    currItem.setTag2(text);
                }else
                    ret.put(currentField, text);
            }

            @Override
            public void handleEndTag(String currentField) {}

            @Override
            public void handleBeginTag(String currentField) {}
        }, _APPLICATION_TAG_);

    parser.startParsing();

    return ret;
}

```

Se puede observar que la clase `NwXmlStandarParserTextHandler` tiene tres metodos preddefinidos, pero unicamente se usa (en este caso) el metodo encargado de leer el texto intermedio, obviando las etiquetas Begin y End.

El metodo `handleText()` se encargara de recorrer el fichero XML, y sera donde se establezcan las etiquetas que se quieren mapear.

```

        if(currentField.equals(_TAG1_TAG_)){
            ret.put(currentField, text);
        }

```

Esta sentencia se encarga de escribir en el mapa:

CLAVE	VALOR
_TAG1_TAG_	tag1

Se puede observar que el objeto “text” es simplemente el valor leído del xml.

```

        if(currentField.equals(_TAG2_TAG_)){
            currItem = new Object();
            ret.put(currentField, currItem);
        }else if(currentField.equals(_TAG21_TAG_)){
            currItem.setTag1(text);
        }else if(currentField.equals(_TAG22_TAG_)){
            currItem.setTag2(text);
        }
    }

```

Y esta otra sentencia se encargara de crear un nuevo objeto y añadirlo al mapa, si se leyese `tag2`, y setearia sus componentes en el caso de leer `tag2_1` o `tag2_2`.

CLAVE	VALOR
_TAG2_TAG_	Object()

Por ultimo, volviendo de nuevo al metodo `parseEjemplo1Data()`, se tomara el mapa resultante del paso anterior, para definir los componentes almacenados en objetos utiles para la aplicación.

Se habla del ultimo metodo `from[Type]Data()`, que siguiendo la linea, se llama `formEjemplo1Data()`

```

        return fromEjemplo1Data(data);
    }

```

Tendrá como entrada el mapa anterior, por tanto, habrá que fijarse en que datos han sido almacenados, que para este ejemplo sería:

CLAVE	VALOR
_TAG1_TAG_	tag1
_TAG2_TAG_	Object()

Por tanto, el método se estructurará de la siguiente manera:

```

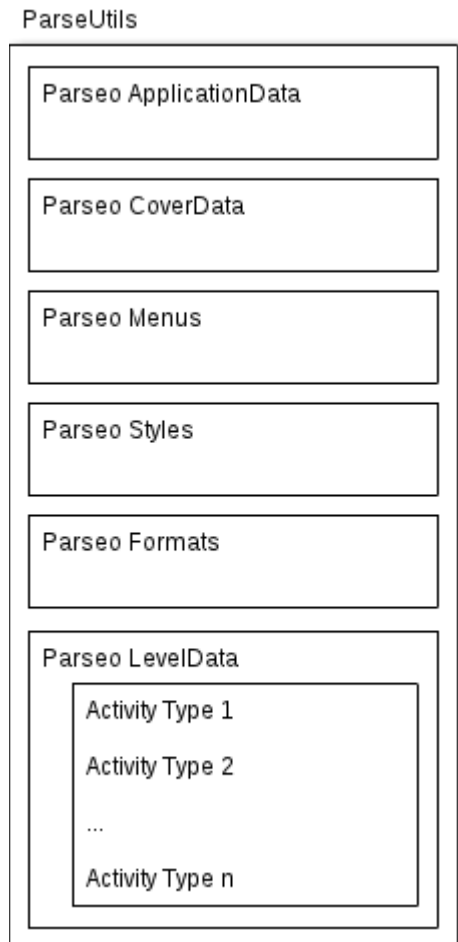
private static Resultado fromEjemplo1Data(Map<String, Object> data) {
    Resultado ret = new Resultado();
    ret.setComp1((String)data.get(_TAG1_TAG_));
    ret.setComp2((Object)data.get(_TAG2_TAG_));
    return ret;
}

```

Como se aprecia, se devolverá una clase `Resultado` con sus correspondientes valores inicializados, de lo recogido del mapa resultante del parseo.

Grupos existentes en la aplicación

Para una mayor organización, el parser esta estructurado en grupos:



Existen por definición, unos bloques que sirven para parsear archivos genéricos, como por ejemplo :

- El fichero referente a Applicationdata, que es la clase que contiene la información general de toda la aplicación. [app.xml]
- El relacionado con la pantalla de portada por defecto. [cover.xml]
- El fichero encargado de definir los estilos de los menús genéricos de la aplicación. [top_menu | bottom_menu | context_menu | sidemenu .xml]
- Los que definen los estilos y formatos para las actividades y niveles de la aplicación. [styles.xml & formats.xml]

Existe también un bloque a parte donde se especifican los parseos para los tipos de pantalla definidos, donde existe un createXpp() y un fromData() globales, y por otro lado un método parse[ActivityType]File() para cada uno de los tipos de actividad (ya que cada pantalla tiene su xml específico con sus propios componentes)

Funcionamiento del EntryPoint

Descripción

El EntryPoint es la pantalla que se carga por primera vez al iniciar la aplicación, después de la pantalla SplashActivity.

Es posible pensar, que exista una pantalla de inicio fija (que es al fin y al cabo una portada), y que sea la misma para todas las aplicaciones.

Pero en ocasiones, el usuario necesita contar con una portada predefinida, o que inicie la aplicación en una pantalla en concreto.

El EntryPoint es el principal encargado de éste cometido.

La función principal es especificar el levelId y el dataId de la pantalla que se quiere iniciar, que como ya se explico antes, corresponde a una pantalla específica.

La forma de establecer dicha información se define en el fichero app.xml, concretamente en el campo entryPoint:

```
<entryPoint>
  <pointLevelId></pointLevelId>
  <pointDataId></pointDataId>
</entryPoint>
```

Simplemente hay que definir el levelId, y su correspondiente dataId, y la aplicación iniciará con la pantalla especificada.

Si no se especifica ningún dato en éste campo, es decir, los campos no tienen ningún contenido establecido, la aplicación carga una pantalla básica por defecto, que no es más que una botonera de inicio.

Algunas características

La pantalla de portada (o como se ha explicado, el EntryPoint), está fuertemente relacionada con los menús, y es por ello, por lo que la pantalla de inicio cuenta con unas restricciones y características únicas, en cuanto a la creación de éstos.

Todos los menús del EntryPoint tienen unas propiedades específicas:

- Las funciones de los menús, no se ven afectadas, por lo que funcionan de igual manera tanto en todas las pantallas, como en la pantalla de EntryPoint.
- No se cuenta con botón **“home”** en ningún tipo de menú para la pantalla de EntryPoint, el sistema lo ignora si el usuario lo hubiese especificado en el xml.
- No se cuenta con botón **“back”** en ningún tipo de menú para la pantalla de EntryPoint, el sistema lo ignora si el usuario lo hubiese especificado en el xml.

Por otro lado se tiene en cuenta si se encuentra en la pantalla de EntryPoint cuando se pulsa el botón “back” del sistema, ya que su función es distinta para el resto de pantallas. Si se encontrase en la pantalla de EntryPoint, el sistema **cerrará la aplicación** si el botón atrás del sistema es pulsado.

Su funcionamiento

El funcionamiento de la pantalla de EntryPoint, se basa únicamente en el paso de parámetros.

Concretamente en una variable booleana que pasa como extra desde los métodos showNextLevel().

Dependiendo desde donde se invoque al método showNextLevel(), ese parámetro podrá ser **true** o **false**.

Por ejemplo, en la clase CoverActivity(), hay que fijarse en que se pasa cómo extra que isEntryPoint es **true**.


```
launchActivity.putExtra(ApplicationData.IS_ENTRY_POINT_TAG, true);
```

En cambio, en la clase AbstractActivityGenerator, isEntryPoint es **false**.

```
launchActivity.putExtra(ApplicationData.IS_ENTRY_POINT_TAG, false);
```

Una vez es pasado como extra, simplemente se recoge su valor en cada Activity, y se pasa posteriormente como parámetro para crear el menú correspondiente.

Visto desde código, en cualquier Activity.

```
IsEntryPoint=  
  (Boolean)intent.getSerializableExtra(ApplicationData.IS_ENTRY_POINT_TAG);  
[. . .]  
createMenus(this, isEntryPoint);
```

Funcionamiento de las Rotaciones

Las rotaciones existen para aportar mas dinamismo o comodidad a la hora de utilizar una aplicación. Hay tipos de pantallas que requieren una mejor vista de sus componentes, y es por ello por lo que se ofrece un sistema de rotaciones personalizado.

Hay que tener en cuenta que rotar una pantalla es simplemente una reestructuración de los componentes, por tanto, ni su número, ni su funcionamiento varía al rotar la pantalla.



Las rotaciones se aplican para todas las pantallas de la aplicación. Por tanto, su propiedad se verá definida en el fichero app.xml.

```
<rotation></rotation>
```

Los valores que puede tomar dicho campo son:

- `portrait`: La pantalla se torna en modo vertical
- `landscape`: La pantalla se torna en modo horizontal
- `both`: Se toman valores del acelerómetro del dispositivo móvil. Por tanto, la pantalla se torna en modo vertical u horizontal dependiendo de la orientación.

Si se insertase un valor desconocido, o vacío, la aplicación definirá la rotación por defecto “both”.

El valor pertenece a la clase `ApplicationData`, pero se necesita setear el tipo de rotación para cada nueva activity. Por ello en cada metodo `onCreate()` de cada activity, se recoge el valor de `ApplicationData`, y se invoca al método `rotateScreen()` de la clase `CreateMenus`, que se encargará de setear el valor para la activity correspondiente con el valor almacenado de la rotación.

```
rotateScreen(this);
```

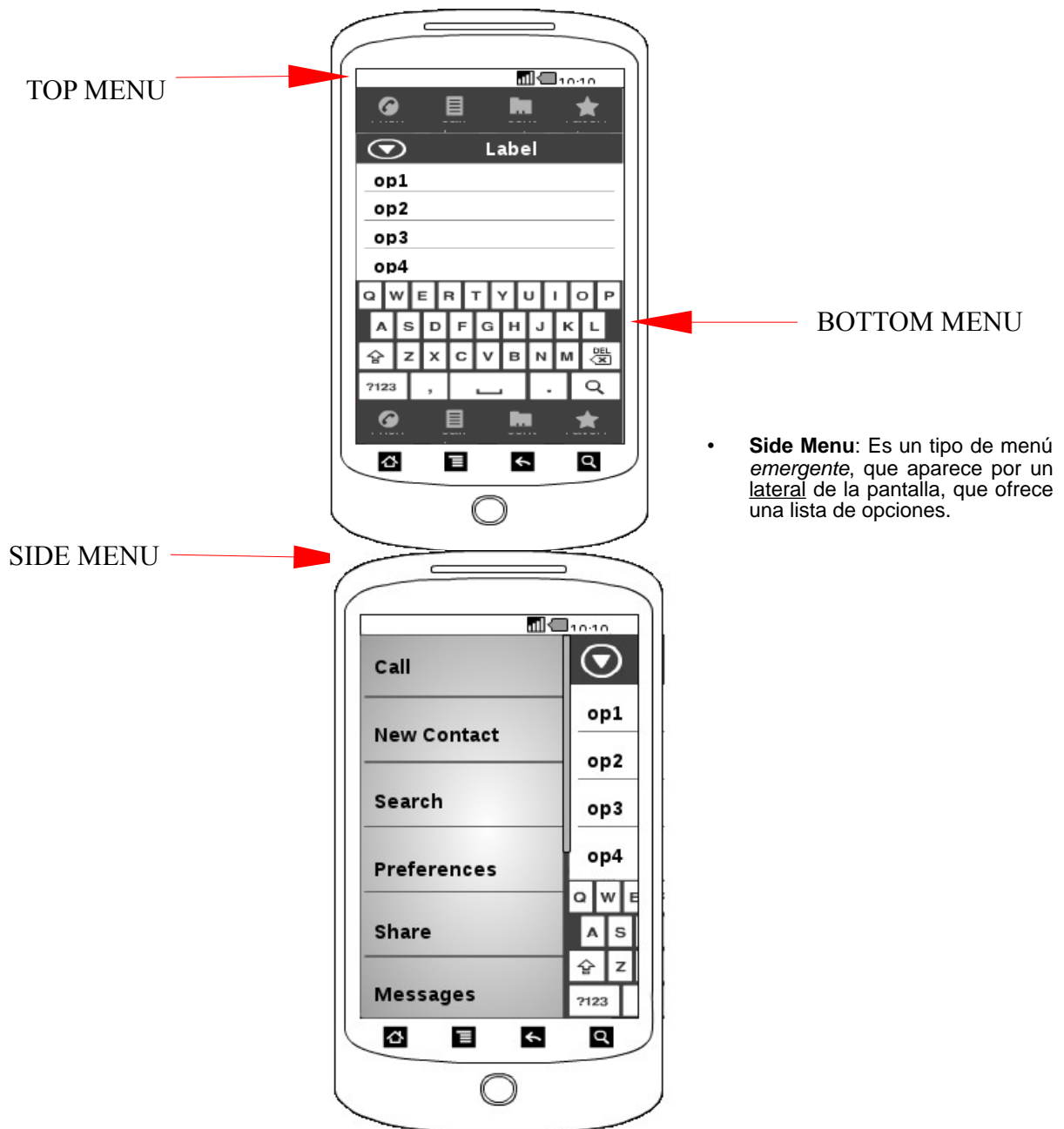
Funcionamiento de los Menús

Los menús en la aplicación

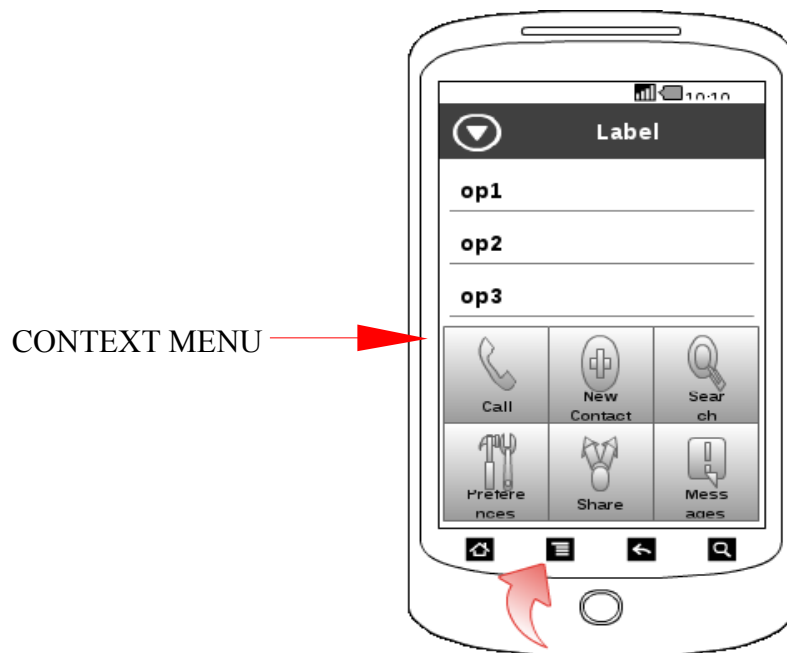
La aplicación cuenta con un sistema de menús, para ofrecer acciones directas en cualquier pantalla.

Existen 4 tipos distintos de menús:

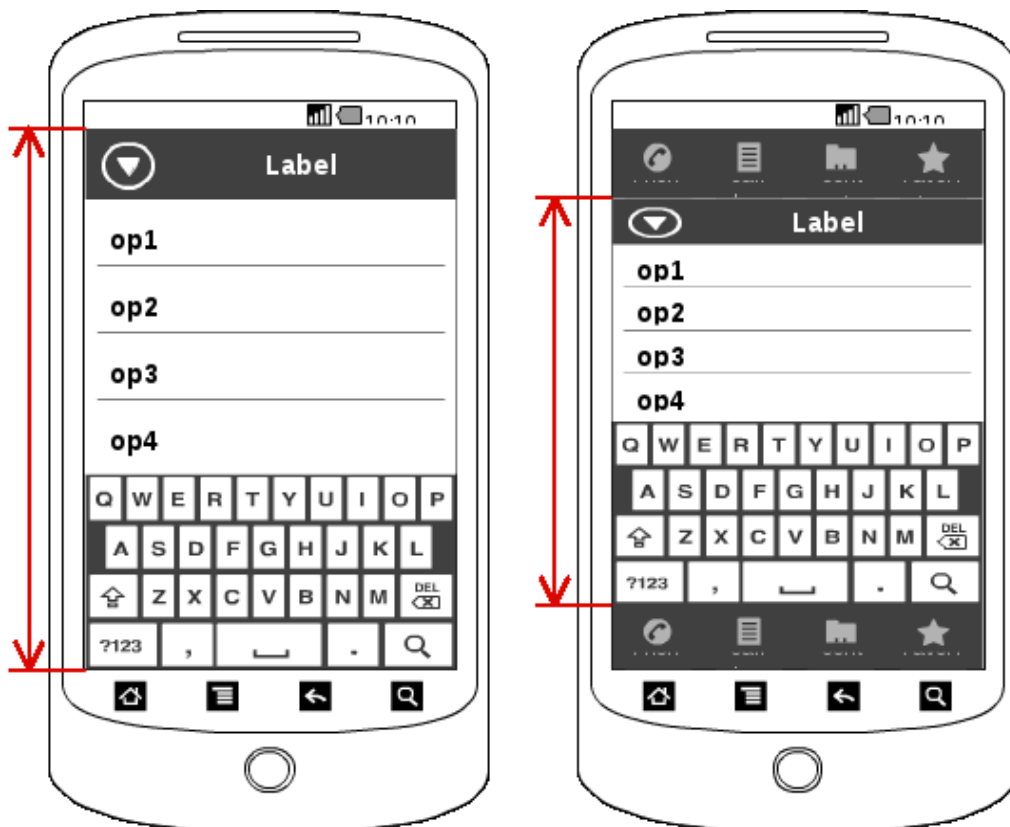
- **Top Menu:** Es un tipo de menú *fijo*, que se posiciona en la parte superior de la pantalla.
- **Bottom Menu:** Es un tipo de menú *fijo*, que se posiciona en la parte inferior de la pantalla.



- **Context Menu:** Es un tipo de menú *emergente*, que aparece al pulsar el boton MENU del sistema.



Los tipos de menú Top Menu y Bottom Menu, son los únicos con propiedad fija, esto quiere decir, que si están definidos, aparecerán fijos en pantalla, reestructurando el resto de componentes de la pantalla. Esos componentes están en un contenedor llamado `contentLayout`, que es precisamente el que reestructura la pantalla en caso de existir alguno de esos tipos de Menú.



En cambio, los tipos de menú Side Menu y Context Menu, al ser emergentes, no influyen en la reestructuración de los componentes de la pantalla. Además, si alguno de estos menús está desplegado, si el botón “back” del sistema es pulsado, su contenido volverá a ocultarse.

Como definir los tipos de Menú

Cada tipo de menú tiene su propio archivo xml, por tanto, al existir 4 tipos distintos de menú, existirán también 4 ficheros xml destinados.

Todos tienen la misma estructura común:

```
<menuActions>
  <action>
    <actionTitle></actionTitle>
    <actionImageName></actionImageName>
    <systemAction></systemAction>
    <leftMargin></leftMargin>
    <widthButton></widthButton>
    <heightButton></heightButton>
    <NextLevel>
      <nextLevelLevelId></nextLevelLevelId>
      <nextLevelDataId></nextLevelDataId>
    </NextLevel>
  </action>
</menuActions>
```

Existe una etiqueta global llamada `menuActions` que engloba toda y cada una de las distintas acciones, con sus correspondientes características.

Una acción (`action`) no es más que la definición para un botón del menú, es decir, es un campo único para cada botón, donde se almacenan los datos tanto de presentación, como de funcionalidad:

- `actionTitle`: nombre de la acción del menú.
- `actionImageName`: nombre de la imagen que presentará ella acción del menú.
- `systemAction`: opción que define si la acción es o no una acción del sistema. Las acciones de sistema son:
 - “back”: acción del sistema para dirigirse a la pantalla anterior.
 - “home”: acción del sistema para dirigirse al EntryPoint.
 - “tts”: acción del sistema para presentar un “Text to Search”
 - “search”: acción del sistema para presentar un “Search”
 - “share”: acción del sistema para presentar un “Share”
 - “map”: acción del sistema para presentar un “Map”
- `leftMargin`: define los márgenes entre las opciones del menú.
- `widthButton`: define el ancho del botón de la acción.
- `heightButton`: define el alto del botón de la acción.
- `NextLevel`: opción que define la pantalla a la que se dirigirá la aplicación al pulsar la opción del menú.

Teniendo en cuenta esta serie de parámetros, hay que tener en cuenta un factor importante.

Cada acción del menú puede o ser una acción de sistema, o abren una nueva actividad, nunca las dos cosas, debido a que las propias acciones del sistema abren también nuevas actividades. Visto en un ejemplo:

Acción de Sistema → Back

```
<menuActions>
  <action>
    <actionTitle>Accion1</actionTitle>
    <actionImageName>img1.png</actionImageName>
    <systemAction>back</systemAction>
    <leftMargin>20</leftMargin>
    <widthButton>40</widthButton>
    <heightButton>40</heightButton>
    <NextLevel>
      <nextLevelLevelId></nextLevelLevelId>
      <nextLevelDataId></nextLevelDataId>
    </NextLevel>
  </action>
</menuActions>
```

Acción normal → Abrir nueva ventana

```
<menuActions>
  <action>
    <actionTitle>Accion2</actionTitle>
    <actionImageName>img2.png</actionImageName>
    <systemAction></systemAction>
    <leftMargin>20</leftMargin>
    <widthButton>40</widthButton>
    <heightButton>40</heightButton>
    <Next Level>
      <nextLevelLevelId>level_1</nextLevelLevelId>
      <nextLevelDataId>data_1_1</nextLevelDataId>
    </Next Level>
  </action>
</menuActions>
```

Se observa por tanto que

- Si el campo `systemAction` está vacío, el campo `NextLevel` toma los valores de una pantalla específica.
- Si el campo `systemAction` tiene valor (válido), el campo `NextLevel` está vacío.

En el caso de que se introduzca algún dato desconocido, por ejemplo, un `systemAction` no definido, la acción simplemente no tendrá ninguna función.

Adición de menús a la aplicación

Una vez se han definido los distintos tipos de menú, hay que asignar a la aplicación los menús que se quieren activar. Ese proceso se centra en el fichero `app.xml`:

```
<menu>
    <topMenu></topMenu>
    <bottomMenu></bottomMenu>
    <contextMenu></contextMenu>
    <sideMenu></sideMenu>
</menu>
```

Simplemente `menu` es un campo en el que se pueden definir si están o no activos los menús explicados anteriormente. En cada una de las 4 etiquetas se asigna el nombre del fichero xml que define un menú. Si no se quiere aplicar algún menú, hay que dejar la etiqueta vacía. Visto en un ejemplo:

```
<menu>
    <topMenu>menu1.xml</topMenu>
    <bottomMenu></bottomMenu>
    <contextMenu>menu2.xml</contextMenu>
    <sideMenu>menu2.xml</sideMenu>
</menu>
```

Esta definición asigna las características explicadas anteriormente:

- Las pantallas tendrán un menú superior con las características especificadas en el fichero `menu1.xml`
- Las pantallas no contarán con ningún menú inferior.
- Las pantallas tendrán un menú contextual con las características especificadas en el fichero `menu2.xml`
- Las pantallas tendrán un menú lateral con las características especificadas en el fichero `menu2.xml`

Cómo se genera un menú en tiempo de ejecución

La clase central que administra todos los menús es `CreateMenus()`, perteneciente al paquete

`com.emobic.android.menu.`

Es en esta clase donde se distribuye el espacio para los distintos menús de la aplicación, donde se generan, y donde se les aplican las funciones a sus acciones.

El método central es `createMenus()`, que tiene como entrada el `activity` actual, y un parametro booleano que indica

si es o no `entryPoint`.

El método se llama en un primer momento al generar una pantalla en cualquier activity:

```
createMenus(this, isEntryPoint);
```

En el metodo se crea un objeto de la clase `ActiveMenus()`, que no es mas que un objeto de ésta misma clase almacenado en `ApplicationData` al parsear el fichero `app.xml`.

Esa clase almacena los nombres asignados a los 4 tipos de menú, por tanto, en el método `createMenus()` se recopila dicha información, y se genera cada uno de los menus, siempre y cuando tenga contenido, si no, asignara el espacio reservado a los componentes de la pantalla.

Por ejemplo:

```
//TOP MENU
if(activeMenus.getTopMenu() != null){
    RelativeLayout topLayout =
        (RelativeLayout)findViewById(R.id.topLayout);
    createCurrentMenu(topLayout, activeMenus.getTopMenu());
}
```

Si el campo de `ActiveMenus` tuviese algun valor, se recogería el espacio predéfinito para el menú, que en este caso es un layout, e invoca al metodo `createCurrentMenu()`.

Es en éste último método donde se rellena ese espacio reservado con la información almacenada en el fichero `xml` designado para ese menú, con cada una de las acciones añadidas.

Particularidades de `ContextMenu` y `SideMenu`:

A diferencia que los menus fijos en pantalla, el `Context Menu` y el `Side Menu`, se diferencian a la hora de plasmar su contenido en pantalla, a pesar de que la funcionalidad se asigna a las acciones de la misma manera.

- **Context Menu:** Al ser llamado por una opcion de sistema, hay que asignar las acciones al propio menu contextual que ofrece Android.

```
contextMenu.add(Menu.NONE, i, Menu.NONE, action.getTitle()).setIcon(idImage);
```

- **Side Menu:** Genera una lista de acciones, y crea una animación para esconder la pantalla actual, y dejar a la vista dicha lista, pudiendo interactuar con ambas vistas.

Temas y Estilos

Los temas y estilos son la base visual en las pantallas. Aportan colores, formas y fondos, que crean una estructura vistosa y agradable al usuario.

La aplicación cuenta con un sistema personalizado de temas y estilos, donde es posible aplicar una temática a un tipo de pantalla, o incluso a una pantalla en concreto.

Procedimiento general

Inicialmente, al parsear el archivo app.xml, se toman dos campos claves: stylesFileName, y formatsFileName.

Cómo su nombre indica, guardarán el nombre de los ficheros XML que contienen los estilos y los formatos.

Una vez hecho eso, se parsean dichos ficheros XML.

- En el XML de formato, se establecen distintos tipos de formato que existen, por ejemplo, title1, title2, text1, etc...

Cada uno de esos tipos de formato contiene información básica a cerca de estilos de textos, como: textColor, textSize, textStyle, typeFace, ...

Tras finalizar ésto, se almacena en un HashMap en un objeto de la clase ApplicationData.

- En el XML de estilos, se asigna a cada tipo de pantalla un background, y además unos campos basicos, referentes a los distintos Views de una pantalla, en los que se indica el formato que va a tener. Estos campos son: **header** (cabecera), **footer** (descripción inferior), **basic_text** (textos simples), **selection_list** (títulos de una lista), **o cualquier otro view NO GENERICO, creado por un usuario.**

Por último, como antes, se almacena en otro HashMap en un objeto de la clase ApplicationData.

- Para aumentar la flexibilidad, el fichero app.xml cuenta con una etiqueta llamada levelFormat, que indica el estilo específico y único para un level de la aplicación, y que actua de igual forma que lo explicado en el punto anterior.

Una vez almacenada toda la información base a cerca de los formatos, ya se pueden asignar a los distintos widgets de una pantalla específica.

Es en los Generators de las distintas pantallas, donde se asignaran los formatos.

Se hace uso del método initializeScreen/initializeScreenWithFormat, encargado de invocar al método initializeWidgetFormat de la clase ApplicationData, que dependiendo de unas características, podrá tener distintas entradas:

- Si al llegar a un level específico, el campo levelFormat tuviese valor (no estuviese completamente vacio), se tendría en cuenta el formato especificado.

```
<levelFormat>
  <backgroundFileName>background.png</backgroundFileName>

  <!-- Generic widgets: header, footer, basic_text, selection_list -->
  <components>header=default;footer=default;</components>
</levelFormat>
```

Se ejecutaría en el ActivityGenerator.java correspondiente:

```
if(isCleanFormat(levelStyle)){
    initializeScreen(activity,"x_ACTIVITY");
}else{
    initializeScreenWithFormat(activity, levelStyle);
}
```

- Si al llegar a un level específico, el campo levelFormat no tuviese valor (estuviese completamente vacío), no se tendría en cuenta el formato especificado, por lo que tendrá como entrada el nombre del tipo de activity (comunmente, el string que hace referencia a dicho tipo).

```
if(isCleanFormat(levelStyle)){
    initializeScreen(activity,"x_ACTIVITY");
}else{
    initializeScreenWithFormat(activity, levelStyle);
}
```

Importancia y versatilidad del método initializeWidgetFormat()

Al método le entran por cabecera dos objetos. Para entender su funcionamiento, hay que centrarse en el segundo, de tipo LevelTypeStyle. Esta clase contiene un pequeño listado, donde se almacenan todos los componentes a los que se quiere aplicar el formato.

Por tanto, dicho método se encargará de recorrer esa lista, para ir aplicando los formatos a cada uno de esos componentes.

Una vez hecho eso, hay que buscar en ApplicationData, en el mapa correspondiente a los FormatStyle, el objeto correspondiente (en este caso el formato) asignado al componente.

El siguiente paso consiste en tomar el View correspondiente al que se quiere aplicar el formato. Si entrase por cabecera una de las cadenas base (header, footer, basic_text, selection_list, ...), el metodo se encargará de tomar el view asociado a ellos:

- header R.id.header
- footer R.id.footer
- basic_text R.id.basic_text
- nombre_de_componente R.id.nombre_de_componente

de manera que se buscaría el View correspondiente a R.id. + "cadena", siempre que exista.

Si no existiese dicho identificador en R.java, se aplicará a todos los textos de la pantalla el formato por defecto de Android.

Cómo crear un nuevo formato

Para crear un nuevo formato, hay que dirigirse al XML correspondiente de formatos. Si no existiese, crear un nuevo fichero XML en la carpeta assets, y escribir su nuevo nombre en el campo

`<formatsFileName></formatsFileName>`

La estructura de este XML es la siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<application>
  <formats>

    <format>
      <name></name>
      <textColor></textColor>
      <textSize></textSize>
      <textStyle></textStyle>
      <typeFace></typeFace>
      <cacheColorHint></cacheColorHint>
      <backgroundSelectionFileName></backgroundSelectionFileName>
    </format>

  </formats>
</application>
```

Descripción de las etiquetas de un formato específico:

- name: nombre del formato
- textColor: color del texto (en hexadecimal). El rango está establecido entre #000000 y #FFFFFF
- textSize: tamaño del texto. Se compone de un número, y una unidad, que puede ser sp, dip, y px.
- textStyle: estilo del texto. Puede ser normal, bold (negrita), e italic (cursiva).
- typeFace: tipo de fuente. Android establece por defecto los tipos normal, sans, serif, y monospace.
- CacheColorHint: color de una lista (en hexadecimal). El rango está establecido entre #00000000 y #FFFFFFF
- backgroundSelectionFileName: nombre del archivo del fondo de cada opción de la lista.

Si se hubiese descargado una nueva fuente, no preddefinida en android, hay que agregarla a la carpeta **assets/fonts**.

Para usarla simplemente hay que asignar al campo typeFace el nombre de la nueva fuente.

Por ejemplo, si se ha descargado la fuente Ubuntu-Bold.ttf, habrá que asignar al campo:

`<typeFace>Ubuntu-Bold</typeFace>`

Por último, conviene tener siempre un formato llamado default, en el que se asignen características básicas.

Como asignar un formato en un tipo de pantalla

Para asignar un formato a un componente de un tipo de pantalla, hay que dirigirse al XML correspondiente de estilos. Si no existiese, crear un nuevo fichero XML en la carpeta assets, y escribir su nuevo nombre en el campo

`<stylesFileName></stylesFileName>`

La estructura de este XML es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
  <styles>

    <type>
      <typeId>x_ACTIVITY</typeId>
      <backgroundFileName></backgroundFileName>
      <components>nombre_componente=nombre_formato;...</components>
    </type>

  </styles>
</application>
```

Descripción de las etiquetas de un formato específico:

- typeId: nombre del tipo de pantalla
- backgroundFileName: nombre del archivo imagen del fondo
- components: es la parte más importante. En este campo se define el formato para cada uno de los tipos de componentes. La sintaxis es como la que se ve en el ejemplo, una composición de:

nombre_componente + "=" + nombre_formato + ";"

Es importante que se siga esa estructura, y que no se añadan caracteres no soportados como saltos de línea, o espacios.

Correcto

```
<components>header=title1;selection_list=list1;</components>
```

Incorrecto

```
<components>
  header=title1;
  selection_list=list1;
</components>
```

En los campos específicos para un componente, simplemente hay que asignarle el nombre de un formato existente (si no existiese, se tomaría el formato por defecto de Android).

En el fichero app.xml, se sigue la misma sintaxis.

Hay que entender que, el formato especificado sobre un level concreto, prevalece sobre el genérico para un tipo de pantalla.

Lo que quiere decir que, si en el fichero XML propuesto para estilos, se establecen unos parámetros para un tipo de pantalla, si para un level de ese mismo tipo de pantalla también se han establecido otros parámetros distintos, la aplicación solo tendrá en cuenta estos últimos.

Descripción de Todos los Tipo de Actividades

Pantalla de Splash	Es la primera pantalla que se muestra de nuestra aplicación. Se muestra para indicar que la aplicación se está cargando. Una vez que desaparece no se puede volver a ella
Pantalla de Cover	Es la pantalla portada. En ella podemos mostrar a todas las opciones y acciones que nuestra aplicación puede soportar.
Pantalla de Galería de Imágenes	Esta pantalla muestra una galería de imágenes al usuario
Pantalla de PDF	Esta pantalla muestra al usuario un lector de pdf
Pantalla QR	Esta pantalla permite al usuario escanear mediante la cámara un código QR
Pantalla de Explorador Web	Esta pantalla muestra al usuario un contenedor Web donde poder visualizar contenido on-line como páginas Web o HTML locales.
Pantalla de Lista	Esta pantalla le muestra al usuario una lista donde. Permite seleccionar entre las opciones que muestra. Contamos con varios formatos de la misma pantalla: <ul style="list-style-type: none">• Lista sólo con texto• Lista con imagen junto al texto
Pantalla de Video	Esta pantalla permite al usuario visualizar un video. El video puede estar en el propio dispositivo o puede ser on-line
Pantalla de Mapa	Esta pantalla muestra al usuario un mapa dependiendo de su localización. El mapa puede contener marcas de sitios preferidos que el usuario puede añadir
Pantalla de Búsqueda	Esta pantalla permite al usuario buscar una porción de texto, georeferencia o imagen dentro del framework
Pantalla Formulario	Esta pantalla permite al usuario crear una nueva pantalla simplemente rellenando los campos necesarios
Pantalla Imagen + Texto	Esta pantalla muestra al usuario una Imagen acompañado por un texto
Pantalla de Imagen con Zoom	Esta pantalla permite mostrar al usuario una imagen aumentada
Pantalla de Imagen + Lista	Esta pantalla nos muestra una lista seleccionable por el usuario, acompañada por una imagen ajena a las celdas.
Pantalla de Multimedia	Esta pantalla permite mostrar al usuario un menú para acceder a las acciones multimedia como pueden ser: <ul style="list-style-type: none">• Acceso a Fotos• Acción de Compartir• Acceso a Videos• Acceso a grabaciones de Voz• etc ...
Pantalla de Calendario	Esta pantalla muestra al usuario un calendario ajeno al calendario del sistema. Este calendario muestra los eventos que la aplicación internamente quiera mostrar.
Pantalla de Quiz	Permite generar tests con varias respuestas y muestra los resultados. También permite generar aventuras interactivas con varios finales que dependen de las decisiones tomadas a lo largo de las preguntas.

Ventana de Audio

Descripción

La ventana de Audio permite reproducir audio, tanto local como externo.

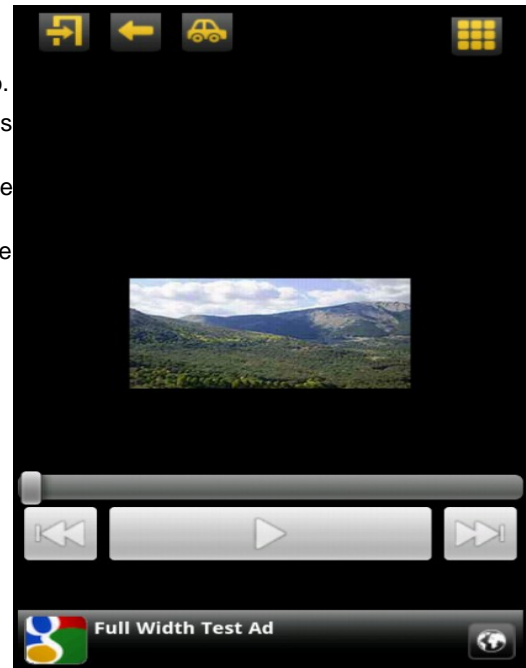
La reproducción de un archivo externo se produce en straming. Es posible reproducir el audio mientras se descarga en segundo plano.

La ventana también muestra información y una imagen de la pista de audio.

Si se va a reproducir una canción, es posible añadir las letras para que aparezcan en la ventana mientras se reproducen.

Incluye un panel de controles básico para la reproducción de audio:

- Botón Play/Pause
- Botón Anterior
- Botón Siguiente



Ejemplo de ventana básica de Audio

Clases y archivos utilizados

Nombre de la clase	Descripción
AudioLevelDataItem	Almacena todos los datos de una ventana de audio
ImageListActivity	Actividad principal de audio. Se reutiliza ImageListActivity.
AudioActivityGenerator	Generador de la actividad de audio.
audio_layout.xml	Layout de la ventana de audio.

Permisos necesarios

Permiso	Descripción
android.permission.INTERNET	Necesario para descargar el audio

Funcionamiento / Generator

La ventana de Audio tiene los siguientes View en su layout:

```

item = (AudioLevelDataItem)data.findByNextLevel(Next Level);
audioBar = (SeekBar) activity.findViewById(R.id.seekBarAudio);
playPause = (ImageButton) activity.findViewById(R.id.playPauseButton);
prev = (ImageButton) activity.findViewById(R.id.prevButton);
next = (ImageButton) activity.findViewById(R.id.nextButton);
mp = new MediaPlayer();
handler = new Handler();

```

La reproducción de audio se realiza mediante el View MediaPlayer de Android (mp). La barra de progreso es un seekBar (audioBar) que utiliza un Handler para manejar las actualizaciones (handler).

Hay botones para el control de la pista (playPause, prev y next).

El funcionamiento y los métodos disponibles para el MediaPlayer pueden consultarse desde Android Developer: <http://developer.android.com/reference/android/media/MediaPlayer.html>.

A continuación se muestra cómo funciona la ventana y cómo hace uso del media player para reproducir contenidos de audio. Toda la lógica de la ventana está dentro del método *loadAppLevelData*, en la clase *AudioActivityGenerator*. Desde ahí se configura el layout con la información del *item* del tipo *AudioLevelDataItem*, así como la creación de de los banners.

```
initializeHeader(activity, item);

//Create Banner
CreateMenus c = (CreateMenus)activity;
c.createBanner();

//Audio Image
ImageView audiolImage = (ImageView) activity.findViewById(R.id.imageAudio);
if (item.getImage()!=null){
    try {
        audiolImage.setImageDrawable(ImagesUtils.getDrawable(activity, item.getImage()));
    } catch (InvalidFileException e1) {
        e1.printStackTrace();
    }
}

//Description
TextView description = (TextView) activity.findViewById(R.id.description);

//Lyrics
if (item.getLyrics()!=null){
    TextView lyrics = (TextView) activity.findViewById(R.id.lyrics);
    lyrics.setText(item.getLyrics());
}
```


Después viene la configuración más importante, que es la que gestionará la reproducción del mp3 mediante el MediaPlayer.

```
//Media Player
mp.setOnBufferingUpdateListener(new OnBufferingUpdateListener() {
    @Override
    public void onBufferingUpdate(MediaPlayer mp, int percent) {
        audioBar.setSecondaryProgress(percent);
    }
});
mp.setOnCompletionListener(new OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mp) {
        playPause.setImageResource(R.drawable.ic_media_play);
    }
});
mp.setOnPreparedListener(new OnPreparedListener() {
    @Override
    public void onPrepared(MediaPlayer mp) {
        play();
    }
});
```

OnBufferingUpdateListener escucha los cambios en el buffer del MediaPlayer. Permite actualizar la barra secundaria del seekBar (audioBar).

OnCompletionListener se usa para saber cuándo termina la reproducción del audio. Cambia el botón de Play/pause a Play para que se pueda volver a reproducir.

OnPreparedListener se utiliza para comprobar cuándo está preparado el MediaPlayer para reproducirse. Es importante para las reproducciones en streaming porque se necesita un periodo inicial de descarga de audio.

```
[...]

//Button Play/Pause
playPause.setImageResource(R.drawable.ic_media_play);
playPause.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        playPause();
    }

});

[...]
```

```
/**
 * Method for play/pause button. If is playing, pause. Else, play.
 */
private void playPause(){
    if (firstTime){
        try {
            mp.setDataSource(item.getAudioUrl());
        } catch (IllegalArgumentException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SecurityException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IllegalStateException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        mp.prepareAsync();
        firstTime = false;
    }else{
        if(mp.isPlaying()){
            pause();
        }else{
            play();
        }
    }
}
```

El botón play/pause simplemente llama a un método que comprueba si hay que pausar o reanudar la reproducción.

Se ha incluido la comprobación de “primera vez” para, solo en ese caso, inicializar el MediaPlayer con la url obtenida y que comience a descargar el audio en segundo plano hasta que sea reproducible.

Los métodos play y pause se encargan de reproducir/reanudar y pausar la reproducción respectivamente. Además, también cambian el icono del botón play/pause al correspondiente según la situación.

```
/**
 * Play music
 */
private void play(){
    mp.start();
    playPause.setImageResource(R.drawable.ic_media_pause);
    primarySeekBarProgressUpdater();
}
/**
 * Pause music
 */
private void pause(){
    mp.pause();
    playPause.setImageResource(R.drawable.ic_media_play);
    primarySeekBarProgressUpdater();
}
```

Por último, el método *primarySeekBarProgressUpdater* se encarga de actualizar la seekBar en la posición correspondiente al tiempo de reproducción.

```
/**
 * Updates primary bar progress
 */
private void primarySeekBarProgressUpdater() {
    audioBar.setProgress(((int)(((float)mp.getCurrentPosition()/mp.getDuration())*100)); // This
    math construction give a percentage of "was playing"/"song length"
    if (mp.isPlaying()) {
        Runnable notification = new Runnable() {
            public void run() {
                if (mp.isPlaying()){
                    primarySeekBarProgressUpdater();
                }
            }
        };
        handler.postDelayed(notification,1000);
    }
}
```

Parte de este código ha sido recopilado desde el siguiente enlace:

<http://www.hrpin.com/2011/02/example-of-streaming-mp3-mediafile-with-android-mediaplayer-class>

Ventana de Calendario

Descripción

La ventana de calendario permite mostrar eventos en un calendario de aspecto mensual. Cada evento tiene asociada una descripción y puede llevar a una nueva ventana.

En el calendario se indican los eventos con colores en los días de la semana:

- **Azul:** Indica que existen eventos en el día
- **Círculo rojo:** Indica el día actual.

Cuando se pulse un día del calendario se mostrarán los eventos (si los hay) en la tabla de eventos. Es posible cambiar el mes para visualizar los posibles eventos en otros meses próximos y pasados.



Ejemplo de ventana básica de calendario

Clases y archivos utilizados

Nombre de la clase	Descripción
CalendarLevelDataItem	Almacena todos los datos de una ventana de calendario
EventDataItem	Almacena los datos de un evento
CalendarActivity	Actividad principal del calendario
CalendarActivityGenerator	Generador de la actividad de Calendario
calendar_layout.xml	Layouts par la ventana de calendario (landscape/normal).
values/dimens.xml	Contiene las dimensiones necesarias para mostrar CalendarView
drawable/typeb_calendar_today.png	Drawable para indicar el día actual en el calendario.
drawable/calendar_week_es.png	Drawable de los días de la semana para CalendarView
values/strings.xml	En strings se almacenan los días de la semana utilizados por CalendarHelper.
CalendarHelper	Clase para gestionar los meses. Devuelve el mes desde strings en R
CalendarView	View de un calendario mensual.
Cell	Clase necesaria para mostrar celdas en el CalendarView

Permisos necesarios

Permiso	Descripción
<i>Sin permisos especiales</i>	

Funcionamiento / Generator

El calendario se basa principalmente en CalendarView. (<http://code.google.com/p/android-calendar-view/>).

Es un View que muestra un calendario en una cuadrícula. El método initCells() es el que se encarga de inicializar el calendario y dibujar las celdas con los días correspondientes al mes. Detecta si es el día actual o si el día contiene eventos para señalizarlo.

```
// paint days with events
// Searching events with this day
if (events!=null){
    String dateKey;
    int dayNumber = tmp[week][day].day;
    if (dayNumber<10){
        dateKey = "0"+dayNumber;

    }else{
        dateKey = ""+dayNumber;
    }
    int monthNumber = mHelper.getMonth()+1;
    if (monthNumber <10){
        dateKey = dateKey+"/0"+monthNumber ;
    }else{
        dateKey = dateKey+"/"+monthNumber;
    }
    int yearNumber = mHelper.getYear();
    dateKey = dateKey+"/"+yearNumber;
    if(events.get(dateKey)!=null) {
        mCells[week][day] = new EventCell(tmp[week][day].day, new Rect(Bound),
            CELL_TEXT_SIZE);
    }
}

Bound.offset(CELL_WIDTH, 0); // move to next column

// get today
if(tmp[week][day].day==thisDay && tmp[week][day].thisMonth) {
    mToday = mCells[week][day];
    mDecoration.setBounds(mToday.getBound());
}
```

Desde CalendarActivity se gestionan las pulsaciones en un día concreto desde el método onTouch(). Por cada pulsación se comprueba si el día tiene eventos para actualizar la lista.

```
//Update the textView
TextView tv = (TextView) findViewById(R.id.textViewDayEvents);
tv.setText(getResources().getString(R.string.events_day)+dateKey);
try {
    ApplicationData applicationData = ApplicationData.readApplicationData(this);
    if(applicationData != null){
        Intent intent = getIntent();
        NextLevel nextLevel =
            (NextLevel)intent.getSerializableExtra(ApplicationData.NEXT_LEVEL_TAG);
        CalendarActivityGenerator generator = (CalendarActivityGenerator)
            applicationData.getFromNextLevel(this, nextLevel);
        generator.updateList(this, dateKey);
    }
} catch (InvalidFileException e) {
    Log.e("CalendartActivity", e.getLocalizedMessage());
    Toast.makeText(getBaseContext(), e.getMessage(), Toast.LENGTH_SHORT).show();
}
```

Desde CalendarActivityGenerator se configuran los botones para cambiar de mes y se gestiona la lista de eventos mediante el método updateList. (figura 2)

Para actualizar el mes en el CalendarView se utilizan los métodos `previousMonth()` y `nextMonth()`. H es un CalendarHelper que permite devolver el mes en String a partir de un número. Configura el cuadro de texto que aparece encima del calendario con el nombre del mes que es visible. (ver figura 1).

```
final CalendarHelper h = new CalendarHelper(activity);
Button prev = (Button) activity.findViewById(R.id.prevMonthButton);
Button next = (Button) activity.findViewById(R.id.nextMonthButton);
prev.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        cv.previousMonth();
        tv.setText(h.getMonthInString(cv.getMonth()+1));
    }
});
next.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        cv.nextMonth();
        tv.setText(h.getMonthInString(cv.getMonth()+1));
    }
});
```

Figura 1. Configuración de los botones de mes próximo y anterior en CalendarActivityGenerator.

```
public void updateList(Activity activity, String date){
    AppLevelData data = appLevel.getData(activity);
    CalendarLevelDataItem item = (CalendarLevelDataItem)data.findByNameLevel(Next Level);
    HashMap<String,TreeSet<EventDataItem>> TotalEvents = item.getEvents();
    TreeSet<EventDataItem> eventsInDate = TotalEvents.get(date);
    ArrayList<EventDataItem> eventsList = new ArrayList<EventDataItem>();
    //Maybe null
    if (eventsInDate!=null){
        eventsList.addAll(eventsInDate);
    }

    //Putting the list in the ListView
    ListView lv = (ListView)activity.findViewById(R.id.listViewCalendarEvents);
    lv.setAdapter(new CallListAdapter(activity, R.layout.list_item, eventsList));
    lv.setTextFilterEnabled(true);
}
```

Figura 2. Método para actualizar la lista de eventos desde CalendarActivityGenerator.

Notas

- Para añadir la ventana del calendario se ha utilizado el código del proyecto android-calendar-view: <http://code.google.com/p/android-calendar-view/>

- El CalendarView descargado devuelve el número del mes -1 (Julio=6, Mayo=4...). Hay que incrementarlo para que coincida con la nomenclatura utilizada en el resto de la aplicación. Esto implica cambiarlo tanto en el método loadAppLevelData de CalendarActivityGenerator como en CalendarActivity.
- Se han añadido Strings a strings.xml correspondientes a los meses del año. month<numeroMes> para poder soportar diferentes idiomas.
- Clase de control CalendarHelper que ayuda a gestionar algunas operaciones del calendario.

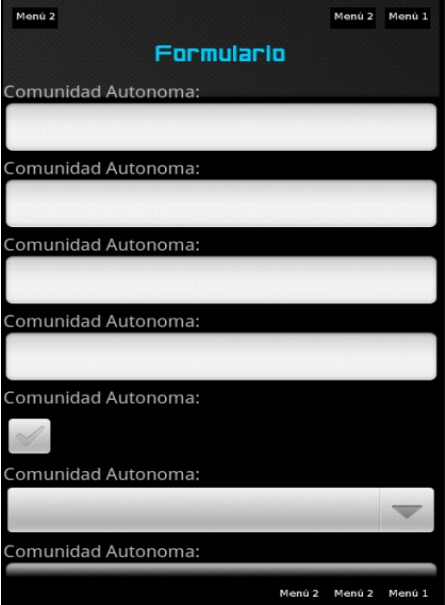
Ventana de formulario

Descripción

La ventana de formulario se utiliza para mostrar formularios con campos para que el usuario los pueda rellenar.

Los campos que se mostrarán son previamente definidos por el xml y se crearán en tiempo de ejecución.

Incluye la funcionalidad de crear aplicaciones dinámicamente al pulsar el botón de aceptar. Puede utilizarse para mostrar diferentes pantallas según los datos introducidos por el usuario.

The screenshot shows a mobile application interface with a dark background. At the top, there are three menu items: 'Menú 2', 'Menú 2', and 'Menú 1'. Below them is a title 'Formulario' in red. The main area contains six input fields, each preceded by the text 'Comunidad Autonoma:'. The first five are standard text input fields. The sixth is a dropdown menu with a checkmark icon on the left and a downward arrow on the right. At the bottom right, there is a button with a right-pointing arrow. The bottom of the screen shows the same three menu items: 'Menú 2', 'Menú 2', and 'Menú 1'.

Ejemplo de ventana básica formulario

Clases y archivos utilizados

Nombre de la clase	Descripción
FormLevelDataItem	Almacena todos los datos de una ventana de formulario
FormDataItem	Almacena los datos de un campo de un formulario
FormFieldType	Tipos de
ImageListActivity	Actividad principal de la ventana de formulario. Se reutiliza ImageListActivity
FormActivityGenerator	Generador de la actividad de la ventana formulario
Form.xml	Layout para la ventana de formulario

Permisos necesarios

Permiso	Descripción
<code>android.permission.INTERNET</code>	Necesario para poder descargar nuevos fragmentos de la aplicación al pulsar el botón de enviar

Funcionamiento / Generator

Toda la lógica del funcionamiento de la ventana de formularios está dentro de FormActivityGenerator.

Antes de iniciar la ventana el layout del formulario (formLayout) está vacío. Desde loadAppLevelData se cargan los datos de FormLevelDataItem y se configura el layout con los campos que sean hayan sido declarados.

```
LinearLayout formLayout = (LinearLayout)activity.findViewById(R.id.formLayout);
for (FormDataItem dataItem : item.getList()){
    TextView label = new TextView(activity);
    label.setText(dataItem.getFieldLabel());
    formLayout.addView(label);
    insertField(activity, dataItem, formLayout);
}
```

La lista item.getList() contiene todos los campos que tiene el formulario. El bucle for los va creando y añadiendo al layout del formulario.

El label es el texto que aparece describiendo el campo. El campo es el Field y puede ser de varios tipos. Para crear un componente específico por cada tipo de campo se utiliza el método insertField.



```
private void insertField(Activity activity, FormDataItem dataItem, LinearLayout formLayout) {
    View control = null;

    switch (dataItem.getType()) {
        case INPUT_TEXT:
            control = insertTextField(activity, dataItem);
            break;
        case INPUT_NUMBER:
            control = insertNumberField(activity, dataItem);
            break;
        [...]
        default:
            break;
    }

    formLayout.addView(control);
    controlsMap.put(dataItem.getFieldName(), control);
}
```

En el método insertField() se comprueba de qué tipo es el campo del formulario para crear el View específico y añadirlo al layout del formulario. Para poder acceder fácilmente a los Views creados se utiliza controlsMap, declarado previamente. Es un mapa Hash donde se guardan los View con clave el nombre del campo.

Cada tipo de campo tiene su propio constructor de View específico. El nombre de estos constructores son insert*Field().

Por ejemplo, el constructor del View para contraseña:

```
private EditText insertPasswordField(Activity activity, FormDataItem dataItem) {  
    EditText txt = insertTextField(activity, dataItem);  
    txt.setInputType(InputType.TYPE_CLASS_TEXT  
InputType.TYPE_TEXT_VARIATION_PASSWORD);  
    return txt;  
}
```

Por último, para terminar la creación del layout de formulario, se añade la acción de enviar los datos a un botón de enviar.

```
Button submit = new Button(activity);  
submit.setText(R.string.form_submit_button);  
submit.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        processSubmit(activity);  
    }  
});  
formLayout.addView(submit);
```

ProcessSubmit es el método encargado de enviar una petición HTTP con los datos del formulario, y a partir de la información recibida, reconstruir el ApplicationData con las nuevas posibles ventanas (levels).

Para recuperar todos los datos introducidos por el usuario se utiliza el controlsMap donde se han almacenado todos los View. Todo se hace con el método createParameters().

```
private List<NameValuePair> createParameters() throws RequiredFieldException {  
    List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(item.getList().size());  
  
    for(FormDataItem dataItem : item.getList()){  
        View view = controlsMap.get(dataItem.getFieldName());  
        final String paramValue = getControlText(view, dataItem);  
  
        if(paramValue != null && paramValue.length() > 0){  
            nameValuePairs.add(new BasicNameValuePair(dataItem.getFieldName(), paramValue));  
        }else{  
            if(dataItem.isRequired())  
                throw new RequiredFieldException(dataItem);  
        }  
    }  
    return nameValuePairs;  
}
```

Se recuperan los datos a partir del nombre del campo y se almacenando en un par nombre, valor y se van añadiendo a una lista que contendrá todos los valores. Sin embargo, los datos a recuperar difieren según el tipo de View que tenga el campo. No se recuperan igual los datos de un cuadro de texto que de un picker de fecha. Para realizar esta recuperación se utiliza el método `getControlText` que transforma el contenido considerado importante de un View en un string.

```
private String getControlText(View view, FormDataItem dataItem) {
    String ret = "";
    switch (dataItem.getType()) {
        case INPUT_PASSWORD:
            ret = ((EditText)view).getText().toString();
            break;
        case INPUT_CHECK:
            ret = ((CheckBox)view).isChecked() ? "true":"false";
            break;
        case INPUT_PICKER:
            ret = ((Spinner)view).getSelectedItem().toString();
            break;
        [...]
        default:
            break;
    }
    return ret;
}
```

La excepción `RequiredFieldException` simplemente se lanza cuando no se ha completado un campo obligatorio.

Una vez se tienen todos los datos recuperados, se `processSubmit` se encarga de enviar la petición HTTP con la lista de valores.

```
parameters = createParameters();

try {
    URL url = new URL(item.getActionUrl());

    HttpClient httpClient =
getHttpClient(url.getProtocol().equalsIgnoreCase(HTTPS_PROTOCOL));
    HttpPost httpPost = new HttpPost(item.getActionUrl());

    try {
        // Add your data
        httpPost.setEntity(new UrlEncodedFormEntity(parameters));
    }
}
```

La respuesta contiene la información que habrá que añadir al ApplicationData utilizando el método mergeAppDataFromString.

```
// Execute HTTP Post Request
```

```
    HttpResponse response = httpClient.execute(httppost);
```

```
    if(response.getStatusLine().getStatusCode() == HttpStatus.SC_OK){
```

```
        String str = EntityUtils.toString(response.getEntity());
```

```
        ApplicationData.mergeAppDataFromString(activity, str);
```

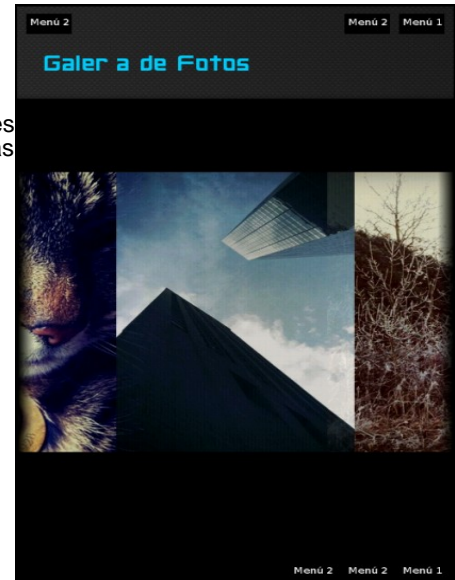
```
        showNextLevel(activity, item.getNextLevel());
```

```
    }
```

Ventana de galería de imágenes

Descripción

La galería de imágenes se utiliza para mostrar un conjunto de imágenes seleccionadas en una sola ventana. El usuario puede desplazarse entre las imágenes para verlas.



Ejemplo de ventana básica de galería

Clases y archivos utilizados

Nombre de la clase	Descripción
ImageGalleryLevelDataItem	Almacena todos los datos de una ventana de galería
ImageDataItem	Almacena la información de una Imagen.
ImageListActivity	Actividad principal de la ventana de galería. Se reutiliza ImageListActivity
ImageGalleryActivityGenerator	Generador de la actividad de la ventana de galería
image_gallery.xml	Layout de de la ventana de galería

Permisos necesarios

Permiso	Descripción
<code>android.permission.INTERNET</code>	Necesario para descargar las imágenes de internet.

Funcionamiento / Generator

La galería de imágenes configura, desde el método `loadAppLevelData` en `GalleryActivityGenerator`, la actividad `ImageListActivity` con el layout de la galería (`image_gallery.xml`).

```
if(hasElements(item.getList())){
    Gallery g = (Gallery) activity.findViewById(R.id.galeria);
    List<Drawable> images = new ArrayList<Drawable>();

    for(ImageDataItem imgItem : item.getList()){
        Drawable drawable = createDrawable(activity, imgItem.getImageFile());
        images.add(drawable);
    }

    g.setAdapter(new ImageAdapter(activity, images));
}
```

Por cada elemento de la lista se genera un drawable para añadirlo a una lista. La galería necesita un adaptador para gestionar la lista de imágenes.

`ImageAdapter` es el adaptador de imágenes personalizado para configurar el tamaño y la posición.

Ventana Imagen con texto

Descripción

La ventana Imagen con descripción es una ventana que muestra una imagen con un cuadro de texto en la parte superior. Incluye además un enlace debajo del texto con un Next Level y puede llevar a otra pantalla.

El cuadro de texto con la descripción es de formato único y admite scroll si el contenido es mayor que el espacio reservado. Si quiere utilizar texto con diferentes formatos puede considerar la pantalla de contenedor web.



Ejemplo de ventana básica de imagen con descripción.

Clases y archivos utilizados

Nombre de la clase	Descripción
ImageTextDescriptionLevelDataItem	Almacena todos los datos de una ventana de imagen con descripción.
ImageTextDescriptionActivity	Actividad de la ventana de imagen con descripción.
ImageTextDescriptionActivityGenerator	Generador de la actividad de la ventana de imagen con descripción.
image_text_descr.xml	Layout de la ventana de imagen con descripción

Permisos necesarios

Permiso	Descripción
android.permission.INTERNET	Necesario por si la imagen está en internet.

Funcionamiento / Generator

Toda el layout de la ventana se configura desde el método loadAppLevelData en ImageTextDescriptionActivityGenerator. Para cargar la imagen se utiliza ImagesUtils por si es una imagen local o externa.

```
if(hasLength(item.getImageFile())){
    Drawable drawable;
    try {
        drawable = ImagesUtils.getDrawable(activity, item.getImageFile());
        ImageView descrImaga = (ImageView)activity.findViewById(R.id.descr_image);
        descrImaga.setImageDrawable(drawable);
    } catch (InvalidFileException e) {
        Log.e("AppCoverData", e.getLocalizedMessage());
        Toast.makeText(activity, e.getMessage(), Toast.LENGTH_SHORT).show();
    }
}
```

La configuración de los TextView de todo el layout se hace a partir de los datos contenidos en el item del tipo ImageTextDescriptionLevelDataItem.

Por último se añade la acción de Next Level al botón de más información.

```
LinearLayout leerMasBtn =(LinearLayout) activity.findViewById(R.id.descr_mas);
leerMasBtn.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        showNextLevel(activity, item.getNextLevel());
    }
});
```


Ventana Imagen zoom

Descripción

Esta pantalla permite que el usuario pueda ver una imagen con gran Zoom, con una gran resolución. Debido a que la imagen es mayor que una imagen que se pueda mostrar en la galería, contamos con un contenedor 'especial' que permita mostrar la imagen en tamaño real y que además le permite al usuario 'navegar' a través de la imagen para poder ver todas sus partes a un tamaño real.

Clases y archivos utilizados

Nombre de la clase	Descripción
ImageLevelDataItem	Almacena todos los datos de una lista
ImageListActivity	Actividad principal de la ventana de imagen zoom. Se reutiliza ImageListActivity
ImageZoomActivityGenerator	Generador de la actividad de la ventana imagen zoom
image_zoom.xml	Layout de una ventana de imagen zoom

Permisos necesarios

Permiso	Descripción
android.permission.INTERNET	Necesario para descargar las imágenes de internet.

Funcionamiento / Generator

El funcionamiento de la ventana simplemente consiste en un View donde se cargará la imagen contenida recuperada desde el path de ImageLevelDataItem. Se utiliza ImagesUtils para controlar si es imagen local o externa.

```
drawable = ImagesUtils.getDrawable(activity, item.getImageFile());
        ImageView planolImage =
        (ImageView)activity.findViewById(R.id.planolImage);
        planolImage.setImageDrawable(drawable);
        TextView head = (TextView)activity.findViewById(R.id.header);
        head.setText(item.getHeaderText());
```

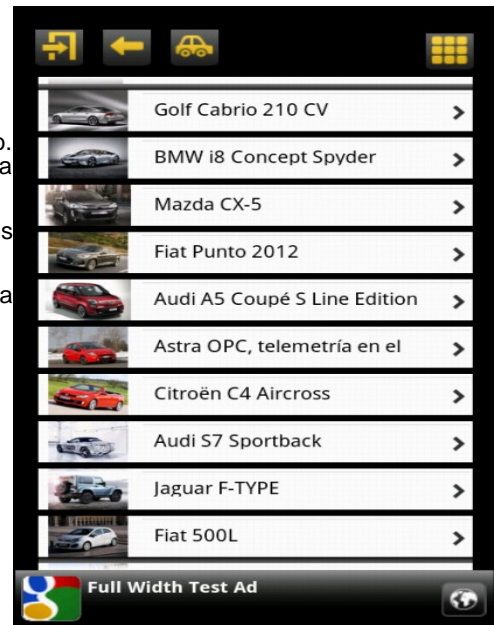
Ventana Lista

Descripción

La ventana de lista muestra una lista con imágenes en cada elemento. Cada elemento de la lista tiene asociado un Next Level y puede llevar a una nueva ventana.

Las imágenes de los elementos de la lista pueden ser tanto locales como remotos.

Si las imágenes son remotas, su carga se realiza de manera asíncrona para agilizar el movimiento y desplazamiento de la lista.



Ejemplo de ventana básica de lista.

Clases y archivos utilizados

Nombre de la clase	Descripción
ListDataItem	Almacena todos los datos de una lista
ImageListActivity	Actividad principal de la ventana de lista
ListActivityGenerator	Generador de la actividad de la ventana de lista
list_item.xml	Layout de un elemento de la lista (texto + imagen)
list_only.xml	Layout para la ventana de la lista.
ImageLoader.java	Clase encargada de gestionar el caché de imágenes
Utils.java	Utilizada por ImageLoader

Permisos necesarios

Permiso	Descripción
<code>android.permission.WRITE_EXTERNAL_STORAGE</code>	Necesario para acceder descargar en caché las imágenes
<code>android.permission.INTERNET</code>	Necesario para descargar las imágenes de internet.

Funcionamiento / Generator

Toda la ventana se gestiona desde ListActivityGenerator. El método loadAppLevelData se encarga de cargar los datos de la lista y añadirlos al ListView.

La gestión de la lista se hace según se explica en Android Developer:

<http://developer.android.com/guide/topics/ui/layout/listview.html>

Sin embargo, para gestionar de manera más eficiente los elementos de la lista, así como la carga en segundo plano en caché de las imágenes de los elementos, se ha utilizado un ListAdapter personalizado. Se han utilizado fragmentos de código y recursos de <https://github.com/thest1/LazyList>, así como de Google ApiDemos.

El nuevo ListAdapter se encargará de acceder al almacenamiento externo para crear una caché con las imágenes descargadas de internet. Además maneja de manera óptima los elementos de la lista.

Dentro del método getView(), cada elemento de la lista (imagen + texto) se almacena en un Holder. El texto es un botón con un Next Level.

```
if(convertView==null){
    vi = inflater.inflate(R.layout.list_item, null);
    holder=new ViewHolder();
    holder.button=button;
    holder.image=(ImageView)vi.findViewById(R.id.list_img);
    vi.setTag(holder);
}
```

Esto se realiza solo cuando convertView es null. De esta manera se evita la carga y actualización continua de los elementos de la lista. Si el holder ya está creado, no es necesario volver a cargarlo.

```
Else
    holder=(ViewHolder)vi.getTag();
```

La configuración de los elementos es importante realizarla fuera de la estructura if/else para que no haya problemas en la visualización de los elementos.

```
View.OnClickListener listener = new View.OnClickListener() {
    public void onClick(View view) {
        showNextLevel(activity, item.getNextLevel());
    }
};

Button button = (Button)vi.findViewById(R.id.list_button);
button.setText(item.getText());
button.setBackgroundResource(R.drawable.list_selector);
button.setOnClickListener(listener);
```

Si la imagen está en internet se utilizará el imageLoader para descargar en segundo plano en la caché las imágenes y mostrarlas cuando estén listas. Si el archivo es local, se buscan utilizando ImagesUtils.

```
if (ImagesUtils.isUrl(item.getImageFile())) {
    holder.image.setTag(item.getImageFile());
    imageLoader.DisplayImage(item.getImageFile(), activity, holder.image);
} else {
    try {
        holder.image.setImageDrawable(ImagesUtils.getDrawable(activity,
item.getImageFile()));
    } catch (InvalidFileException e) {
        e.printStackTrace();
        Toast.makeText(activity, e.getMessage(), Toast.LENGTH_SHORT).show();
    }
}
```

Ventana Lista con Imagen

Descripción

Esta pantalla muestra al usuario una imagen acompañada por una lista. Es parecida a la pantalla de lista. Las características de esta pantalla en concreto es que se va a mostrar una imagen en la parte inferior de la pantalla y que las celdas de la lista sólo van a poder contener texto y no pueden ir acompañadas por otra imagen.



Ejemplo de ventana básica de lista con imágenes.

Clases y archivos utilizados

Nombre de la clase	Descripción
ImageListLevelDataItem	Almacena todos los datos de una lista
ImageListActivity	Actividad principal de la ventana para la lista con imagen
ImageListActivityGenerator	Generador de la actividad de la ventana de lista con imagen
image_list.xml	Layout para la ventana de lista con imagen
image_list_item.xml	Layout para un elemento de la lista (botón)

Permisos necesarios

Permiso	Descripción
<code>android.permission.INTERNET</code>	Necesario para descargar las imágenes de internet.

Funcionamiento / Generator

La configuración del layout se hace desde loadAppLevelData. Se cargan los elementos de la lista, con sus posibles Next Level y la imagen de la parte inferior.

Para cargar la imagen se utiliza ImagesUtils por si es una imagen local o externa.

```
        if (hasLength(item.getImageFile())) {  
Drawable drawable;  
try {  
    drawable = ImagesUtils.getDrawable(activity, item.getImageFile());  
    ImageView descrImaga = (ImageView)activity.findViewById(R.id.descr_image);  
    descrImaga.setImageDrawable(drawable);  
} catch (InvalidFileException e) {  
    Log.e("AppCoverData", e.getLocalizedMessage());  
    Toast.makeText(activity, e.getMessage(), Toast.LENGTH_SHORT).show();  
}  
}
```

La gestión de la lista se hace según se explica en Android Developer:

<http://developer.android.com/guide/topics/ui/layout/listview.html>

```
if (hasElements(item.getList())) {  
    List<ListItemDataItem> objects = item.getList();  
    if (item.isOrder()) {  
        Collections.sort(objects, new Comparator<ListItemDataItem>() {  
            @Override  
            public int compare(ListItemDataItem object1, ListItemDataItem object2) {  
                return object1.getText().compareTo(object2.getText());  
            }  
        });  
    }  
}  
  
ListView lv = (ListView)activity.findViewById(R.id.lv_nivel);  
lv.setAdapter(new NwListAdapter(activity, R.layout.image_list_item, objects));  
lv.setTextFilterEnabled(true);  
}
```

El Adapter para la lista incluye una lista de items y la definición concreta de los tipos de elementos con los estilos de botones y los Next Level, dentro del método getView().

El atributo items contiene los elementos de la lista obtenidos desde ListLevelDataItem.

```
public NwListAdapter(Activity context, int textViewResourceId, List<ListItemDataItem> objects) {  
    super(context, textViewResourceId, objects);  
    this.items = objects;  
    this.activity = context;  
}
```

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    LinearLayout view = (convertView != null) ? (LinearLayout) convertView : createView(parent);
    final ListItemDataItem item = items.get(position);

    View.OnClickListener listener = new View.OnClickListener() {
        public void onClick(View view) {
            showNextLevel(activity, item.getNextLevel());
        }
    };

    Button button = (Button) view.findViewById(R.id.list_button);
    button.setText(item.getText().toUpperCase());
    button.setBackgroundResource(R.drawable.list_selector);
    button.setOnClickListener(listener);

    return view;
}
```

Se recupera el ítem a partir de la posición del elemento marcado, y se añade la función del Next Level a cada botón. Para crear el view se utiliza el layout para los elementos de la lista: image_list_item.xml

```
private LinearLayout createView(ViewGroup parent) {
    LinearLayout item =
        (LinearLayout) activity.getLayoutInflater()
            .inflate(R.layout.image_list_item, parent, false);
    return item;
}
```

Ventana de mapa

Descripción

La pantalla de Mapa muestra un mapa de unas coordenadas específicas, por las que el usuario se puede mover para ver los alrededores a la posición dada.

Además, esta pantalla permite añadir marcas de sitios preferidos o lugares de interés que se cargarán a la vez que se carga el mapa.

Clases y archivos utilizados

Nombre de la clase	Descripción
MapLevelDataItem	Almacena todos los datos de una ventana de Mapa. Incluye una lista de MapDataItemDistance's
MapDataItemDistance	Almacena un MapDataItem + distancia
MapDataItem	Almacena la información de un item que se mostrará en el mapa
MapsActivity	Actividad principal de la ventana para un mapa
MapActivityGenerator	Generador de la actividad de la ventana de mapa
map_screen.xml	Layout de la ventana de mapa

Permisos necesarios

Permiso	Descripción
<code>android.permission.INTERNET</code>	Necesario para utilizar la función de mapas online.
<code>android.permission.ACCESS_FINE_LOCATION</code>	Para geo localización mediante GPS
<code>android.permission.ACCESS_COARSE_LOCATION</code>	Para geo localización mediante redes.

Funcionamiento / Generator

El layout del mapa tiene MapView que se configurará con la información almacenada en MapLevelDataItem. Esta información incluye una lista de sitios de interés entre otra información. La configuración inicial del layout se realiza en el método loadAppLevelData de MapActivityGenerator y la configuración del MapView se realiza con el método loadMapItems.

```
MapView mapView = (MapView) activity.findViewById(R.id.mapview);

        mapView.setBuiltInZoomControls(true);

        List<Overlay> mapOverlays = mapView.getOverlays();

        GeoPoint center = new GeoPoint((int) (CENTER_LATITUDE * 1E6),
                                         (int) (CENTER_LONGITUDE * 1E6));

        Location locationCenter = new Location("Center Point");
        locationCenter.setLatitude(CENTER_LATITUDE);
        locationCenter.setLongitude(CENTER_LONGITUDE);
```

En primer lugar, el mapa se carga en una posición determinada definida en las constantes `CENTER_LATITUDE` y `CENTER_LONGITUDE`. Esta posición inicial está pensada para establecerse con la geolocalización.

Después, por cada sitio de interés (MapDataItem) que hay que mostrar en el mapa, que estará almacenado en MapLevelDataItem, se crea un par <item, distancia> que se almacenará en el atributo nearItems. De esta manera se guardan los lugares de interés que hay que mostrar en el mapa, así como la distancia a la que se encuentran desde el punto inicial del mapa. Además, se añaden al MapView los elementos mediante mapOverlays.


```
nearItems = new ArrayList<MapDataItemDistance>();

for (MapDataItem item : dataItem.getItems()) {
    Drawable drawable = ImagesUtils.getDrawable(activity, item.getIcon());
    itemizedOverlay = new CustomItemizedOverlay(drawable, activity);

    latitudeE6 = item.getLat();
    longitudeE6 = item.getLon();

    point = new GeoPoint((int) (latitudeE6 * 1E6), (int) (longitudeE6 * 1E6));
    overlayitem = new OverlayItem(point, item.getTitle(), item.getAddress());

    itemizedOverlay.addOverlay(overlayitem);
    mapOverlays.add(itemizedOverlay);

    Location itemLocation = new Location(item.getTitle());
    itemLocation.setLatitude(item.getLat());
    itemLocation.setLongitude(item.getLon());
    float distance = locationCenter.distanceTo(itemLocation);
    nearItems.add(new MapDataItemDistance(item, distance));
}
```

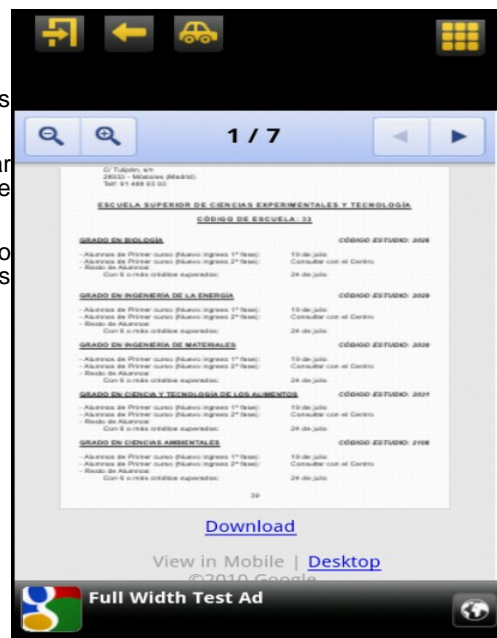
Ventana de PDF

Descripción

La ventana de pdf está pensada para mostrar documentos externos (principalmente de archivos pdf) en la aplicación.

Para documentos externos, la ventana utiliza Google Docs para mostrar el contenido. Además de pdf, es posible utilizar otros formatos de archivos compatibles con Google Docs.

La herramienta de Google Docs permite descargar el archivo para tenerlo localmente y abrirlo con la aplicación por defecto de lectura de archivos pdf que tenga el usuario instalada en su móvil.



Clases y archivos utilizados

Ejemplo de ventana básica de Pdf

Nombre de la clase	Descripción
PdfLevelDataItem	Almacena todos los datos de una ventana de pdf
ImageListActivity	Actividad principal de pdf. Se reutiliza ImageListActivity.
PdfActivityGenerator	Generador de la actividad de pdf.
pdf_activity.xml	Layout de la ventana de pdf.

Permisos necesarios

Permiso	Descripción
<code>android.permission.INTERNET</code>	Necesario para conectarse a internet y cargar los pdfs desde Google Docs Viewer

Funcionamiento / Generator

La ventana de pdf contiene toda su lógica dentro de PdfActivityGenerator. El pdf utilizará Google Docs para mostrar el contenido. Más información sobre Google Docs:

<https://docs.google.com/viewer/>

```
final PdfLevelDataItem item = (PdfLevelDataItem)data.findByNextLevel(NEXT_LEVEL);

if(hasLength(item.getPdfUrl())){
    WebView webview = (WebView) activity.findViewById(R.id.pdfviewer);
    webview.getSettings().setJavaScriptEnabled(true);
    webview.getSettings().setPluginsEnabled(true);
    webview.setWebViewClient(new InsideWebViewClient(activity));
    webview.loadUrl("http://docs.google.com/gview?embedded=true&url="+item.getPdfUrl());
}
```

Dentro del método loadAppLevelData se carga el pdf en la herramienta Google Docs mediante un WebView.

La web que se pasa como parámetro al método loadUrl() del WebView es la dirección del GoogleView seguida del archivo que se quiere leer.

La descarga del documento requiere tiempo y por esa razón se ha incluido una notificación toast para informar que el documento se está descargando. InsideWebClient es un WebClient que se encarga de comprobar el estado de carga

```
/* Class that prevents opening the Browser */
private class InsideWebViewClient extends WebViewClient {
    Context activity;
    ProgressDialog dialog;
    boolean loading;
    public InsideWebViewClient(Context activity){
        super();
        this.activity=activity;
        loading=false;
    }
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        if (Uri.parse(url).getHost().equals("docs.google.com")) {
            // This is my web site, so do not override; let my WebView load the page
            return false;
        }
        // Otherwise, the link is not for a page on my site, so launch another Activity that handles URLs
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        activity.startActivity(intent);
        return true;
    }
    @Override
    public void onPageStarted(WebView view, String url, Bitmap favicon){
        super.onPageStarted(view, url, favicon);
        if (!loading){
            dialog = new ProgressDialog(activity);
            dialog.setMessage(activity.getResources().getString(R.string.cargando_pdf));
            dialog.show();
            loading = true;
        }
        //SHOW LOADING IF IT ISNT ALREADY VISIBLE
    }
    @Override
    public void onPageFinished(WebView view, String url) {
        if (loading){
            dialog.dismiss();
            dialog = null;
        }
    }
}
```

del WebView además de evitar que se inicie el navegador cada vez que se accede a un enlace.

El método sobrescrito `shouldOverrideUrlLoading` es el que comprueba el enlace que se va a cargar pertenece a un dominio concreto o no. En este caso todos los enlaces dentro del dominio de Google Docs mantendrán la carga dentro del WebView. Esto es necesario porque la carga del Google View suele utilizar varios enlaces hasta cargar finalmente. Si no estuviera este método, lo más probable es que se abriera el navegador por defecto del móvil antes de cargar el

documento en el WebView.

```
if (Uri.parse(url).getHost().equals("docs.google.com")) {  
    // This is my web site, so do not override; let my WebView load the page  
    return false;  
}
```

Si el enlace que se está cargando no pertenece al dominio de Google Docs, es o bien porque se ha pulsado el botón descargar (que simplemente es un enlace al lugar donde se aloja el archivo pdf que estamos viendo) o porque se ha pulsado un enlace contenido en el documento pdf. En este caso, se abre el navegador por defecto del móvil para una carga óptima de la dirección web.

```
// Otherwise  
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));  
activity.startActivity(intent);  
return true;
```

Para la gestión de la notificación toast, simplemente se sobrescribe los métodos onPageStarted() y onPageFinished() para iniciar y detener la notificación respectivamente.

Se ha incluido la comprobación de “primera vez” para, solo en ese caso, inicializar el MediaPlayer con la url obtenida y que comience a descargar el audio en segundo plano hasta que sea reproducible.

Los métodos play y pause se encargan de reproducir/reanudar y pausar la reproducción respectivamente. Además, también cambian el icono del botón play/pause al correspondiente según la situación.

```
/**
 * Play music
 */
private void play() {
    mp.start();
    playPause.setImageResource(R.drawable.ic_media_pause);
    primarySeekBarProgressUpdater();
}
/**
 * Pause music
 */
private void pause() {
    mp.pause();
    playPause.setImageResource(R.drawable.ic_media_play);
    primarySeekBarProgressUpdater();
}
```

Por último, el método *primarySeekBarProgressUpdater* se encarga de actualizar la seekBar en la posición correspondiente al tiempo de reproducción.

```
/**
 * Updates primary bar progress
 */
private void primarySeekBarProgressUpdater() {
    audioBar.setProgress((int)
        (((float)mp.getCurrentPosition()/mp.getDuration())*100)); // This math
        construction give a percentage of "was playing"/"song length"
    if (mp.isPlaying()) {
        Runnable notification = new Runnable() {
            public void run() {
                if (mp.isPlaying()) {
                    primarySeekBarProgressUpdater();
                }
            }
        };
        handler.postDelayed(notification,1000);
    }
}
```

Parte de este código ha sido recopilado desde el siguiente enlace:

<http://www.hrpin.com/2011/02/example-of-streaming-mp3-mediafile-with-android-mediaplayer-class>

Ventana de portada

Descripción

La ventana de portada está diseñada para ser la primera ventana de la aplicación (A excepción de SplashActivity). Habitualmente muestra una lista de botones con acciones básicas en la aplicación o enlaces a otras ventanas secundarias.

La ventana de portada básica se compone de una imagen de cabecera y unos botones con Next Level a otras ventanas de la aplicación.

Básicamente es el punto de partida desde donde el usuario puede acceder a todas las funciones de la aplicación.

Es importante saber que toda aplicación tiene que disponer de la Actividad de portada. Es posible modificar la ventana inicial de la aplicación mediante la función entry point (ver entry point), pero la actividad de portada es la que se encarga de gestionarlo (ver más adelante SplashActivity).



Ejemplo de ventana de portada

Clases y archivos utilizados

Nombre de la clase	Descripción
CoverActivity	Actividad principal de la ventana de portada.
CoverActivityGenerator	Generador de la actividad de la ventana web. También se encarga de almacenar los datos de la ventana.
Main.xml	Layout de la ventana de portada.

Permisos necesarios

Permiso	Descripción
<i>Sin permisos especiales</i>	

Funcionamiento / Generator

CoverActivityGenerator
backgroundFileName : String titleFileName : String facebookUrl : String twitterUrl : String wwwUrl : String buttons : List
<<create>> CoverActivityGenerator() getBackgroundFileName() : String setBackgroundFileName(backgroundFileName : String) : void initializeSubActivity(activity : Activity) : void setTitleFileName(titleFileName : String) : void getTitleFileName() : String setButtons(buttons : List) : void getButtons() : List getContentViewResourceId(activity : Activity) : int setFacebookUrl(facebookUrl : String) : void getFacebookUrl() : String setTwitterUrl(twitterUrl : String) : void getTwitterUrl() : String setWwwUrl(wwwUrl : String) : void getWwwUrl() : String

El funcionamiento de la ventana de portada difiere del resto de ventanas del framework. No existe un `LevelDataItem` para la ventana de portada y es el propio Generator el que incluye la información para configurar el layout.

En el diagrama de clases de la clase `CoverActivityGenerator` de la derecha, se puede apreciar que contiene los métodos `get` y `set` propios de cualquier `LevelDataItem`. El parser en `ParseUtils` utilizará los métodos `set`. Más.

También dispone de los atributos donde se guarda la información del layout, como la lista de botones que tendrá la portada.

Por lo tanto, `CoverActivityGenerator` no recupera información de ningún `levelDataItem` y accede a sus propios atributos para configurar el layout desde el método `initializeSubActivity`.

La ventana de portada soporta un máximo de 6 botones, que están declarados previamente en el layout de la portada. Según la información contenida en el xml de la portada, que por lo tanto estará almacenada en `CoverActivityGenerator`, se mostrarán los botones que hayan sido declarados hasta un máximo de 6.

En el método `initializeSubActivity` se comprueba el tamaño de la lista de botones declarados y se van creando uno a uno.


```
if(buttons != null && buttons.size() > 0){
    ImageButton childButton = null;
    if(buttons.size() > 0){
        final AppButton button = buttons.get(0);
        if(button != null){
            childButton = (ImageButton)mainLayout.findViewById(R.id.firstButton);

            childButton.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
                    showNextLevel(activity, button.getNextLevel());
                }
            });
        }
    }
    [...]
    if(buttons.size() > 5) {
        final AppButton button = buttons.get(0);
        if(button != null){
            childButton = (ImageButton)mainLayout.findViewById(R.id.firstButton);

            childButton.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {
                    showNextLevel(activity, button.getNextLevel());
                }
            });
        }
    }
}
```

SplashActivity

La ventana de portada depende de SplashActivity y se crea inmediatamente después. Desde SplashActivity SIEMPRE y a pesar de existir un entry point, se crea la Actividad para la portada. Desde el método

```
Intent mainIntent = new Intent(SplashActivity.this,
                               CoverActivity.class);

SplashActivity.this.startActivity(mainIntent);

SplashActivity.this.finish();
```

Inicio de la actividad CoverActivity desde el método onCreate

Realmente es la ventana de portada (CoverActivity) la que, en el proyecto para Android, gestiona el entryPoint (ver entry point).

Esto se realiza en el método onCreate de CoverActivity:

```
ApplicationData applicationData = SplashActivity.getApplicationData();
    if(applicationData != null){

        /*ENTRY POINT
        * If the field <home> is empty, load the CoverActivity
        * Else, load an Activity with the NextLevel
        */
        setTitle(applicationData.getTitle());

        EntryPoint entryP = applicationData.getEntryPoint();
        String pointLevelId = entryP.getLevelId();
        String pointDataId = entryP.getDataId();

        if(pointLevelId!=null & pointDataId!=null){
            NextLevel nl = new NextLevel (pointLevelId, pointDataId);
            showNextLevel(this, nl);
        }else{
            ActivityGenerator generator = applicationData.getAppCoverData(this);
            generator.initializeActivity(this);
        }

    }
```

La ventana de portada se crea mediante el método *initializeActivity* de *generator*, pero solo en el caso en el que no exista un entry point definido. Si existe, CoverActivity simplemente se encargará de mostrar la nueva ventana a partir del Next Level.

Ventana de QR

Descripción

La pantalla QR es aquella que permite al usuario escanear códigos QR.

Clases y archivos utilizados

Nombre de la clase	Descripción
QrLevelDataItem	Almacena todos los datos de una ventana Qr
QrActivity	Actividad principal de la ventana Qr
QrActivityGenerator	Generador de la actividad de la ventana Qr
qr.xml	Layout de la ventana Qr
IntentIntegrator*	Archivo de librería externa para la lectura de códigos Qr
IntentResult*	Archivo de librería externa para la lectura de códigos Qr

* La librería externa está en el paquete **com.google.zxing.integrator.android**

Permisos necesarios

Permiso	Descripción
<code>android.permission.INTERNET</code>	

Funcionamiento / Generator

El funcionamiento de la ventana Qr se basa en la librería externa para lectura de códigos Qr Zxing.

Puede consultar la documentación del proyecto para comprender su funcionamiento.

<http://code.google.com/p/zxing/>

Ventana de Quiz

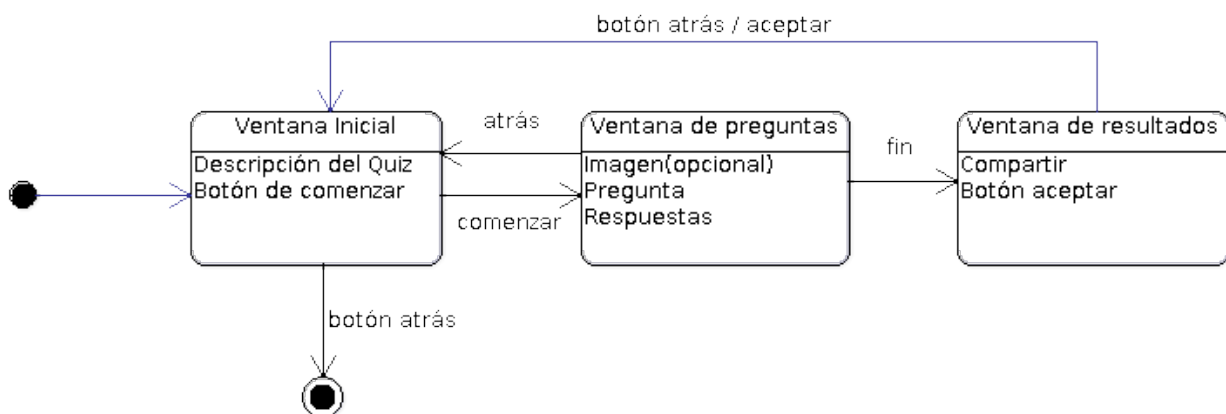
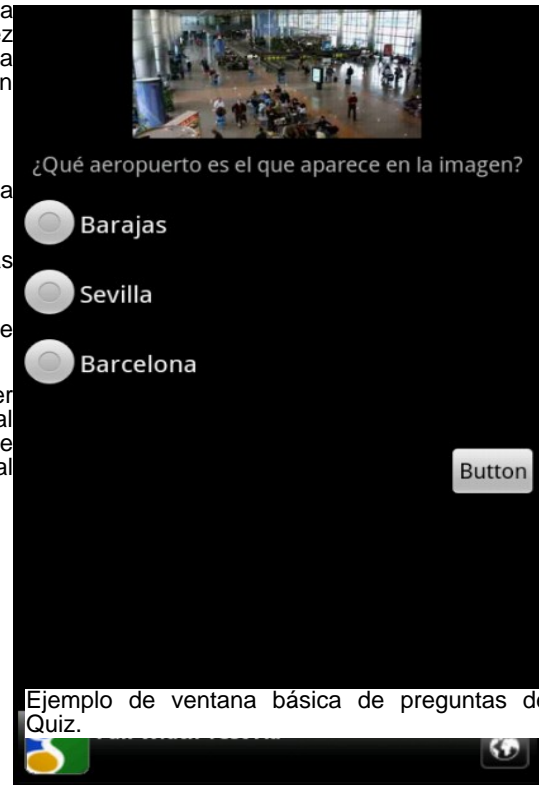
Descripción

La ventana de quiz permite mostrar al usuario un cuestionario con una pregunta, una imagen (opcional) y unas respuestas. Una vez finalizado el quiz se mostrarán los resultados en una nueva ventana según los pesos de las preguntas y las respuestas que se hayan contestado.

La función Quiz dispone varias pantallas:

- Pantalla Inicial: Muestra una descripción del quiz. Es la pantalla de preparación para inicial el quiz.
- Pantalla de preguntas: Muestra las preguntas y respuestas hasta que no existan más.
- Pantalla de resultados: Muestra los resultados después de haber completado el quiz.

Además de los un cuestionario tradicional, la función Quiz puede ser utilizada para crear aventuras. Una aventura es un quiz en el cual cada respuesta tiene definida una nueva pregunta. De esta manera se pueden crear aventuras interactivas en las que el resultado final depende de las decisiones tomadas a lo largo del quiz.



Clases y archivos utilizados

Nombre de la clase	Descripción
QuizLevelDataItem	Almacena todos los datos de una ventana de quiz
QuestionDataItem	Almacena los datos de una pregunta
AnswerDataItem	Almacena los datos de una respuesta
Results	Almacena los resultados de un quiz
QuizActivity	Actividad principal del quiz
QuizQuestionActivity	Actividad para la ventana de preguntas
QuizResultsActivity	Actividad para la ventana de resultados
QuizActivityGenerator	Generador de la actividad de Quiz
quiz_layout.xml	Layout de la ventana inicial del quiz
quiz_questions_layout.xml	Layout de la ventana de preguntas del quiz
quiz_results_layout.xml	Layout de la ventana de resultados del quiz
QuizController	Clase de control para gestionar el quiz

Permisos necesarios

Permiso	Descripción
<i>Sin permisos especiales</i>	

Funcionamiento / Generator

La ventana de Quiz utiliza tanto el la clase *QuizActivityGenerator* como *QuizActivity* para la lógica de los cuestionarios o aventuras.

La configuración básica del layout de la ventana principal se hace desde *QuizActivityGenerator*, en el método *loadAppLevelData*.

```
@Override
protected void loadAppLevelData(Activity activity, AppLevelData data) {
    final QuizLevelDataItem item = (QuizLevelDataItem) data.findByNameLevel(Next Level);

    initializeHeader(activity, item);

    //Create Banner
    CreateMenus c = (CreateMenus)activity;
    c.createBanner();
    ((QuizActivity)activity).setQuiz(item);
    TextView description = (TextView) activity.findViewById(R.id.quizDescription);
    description.setText(item.getDescription());
}
```

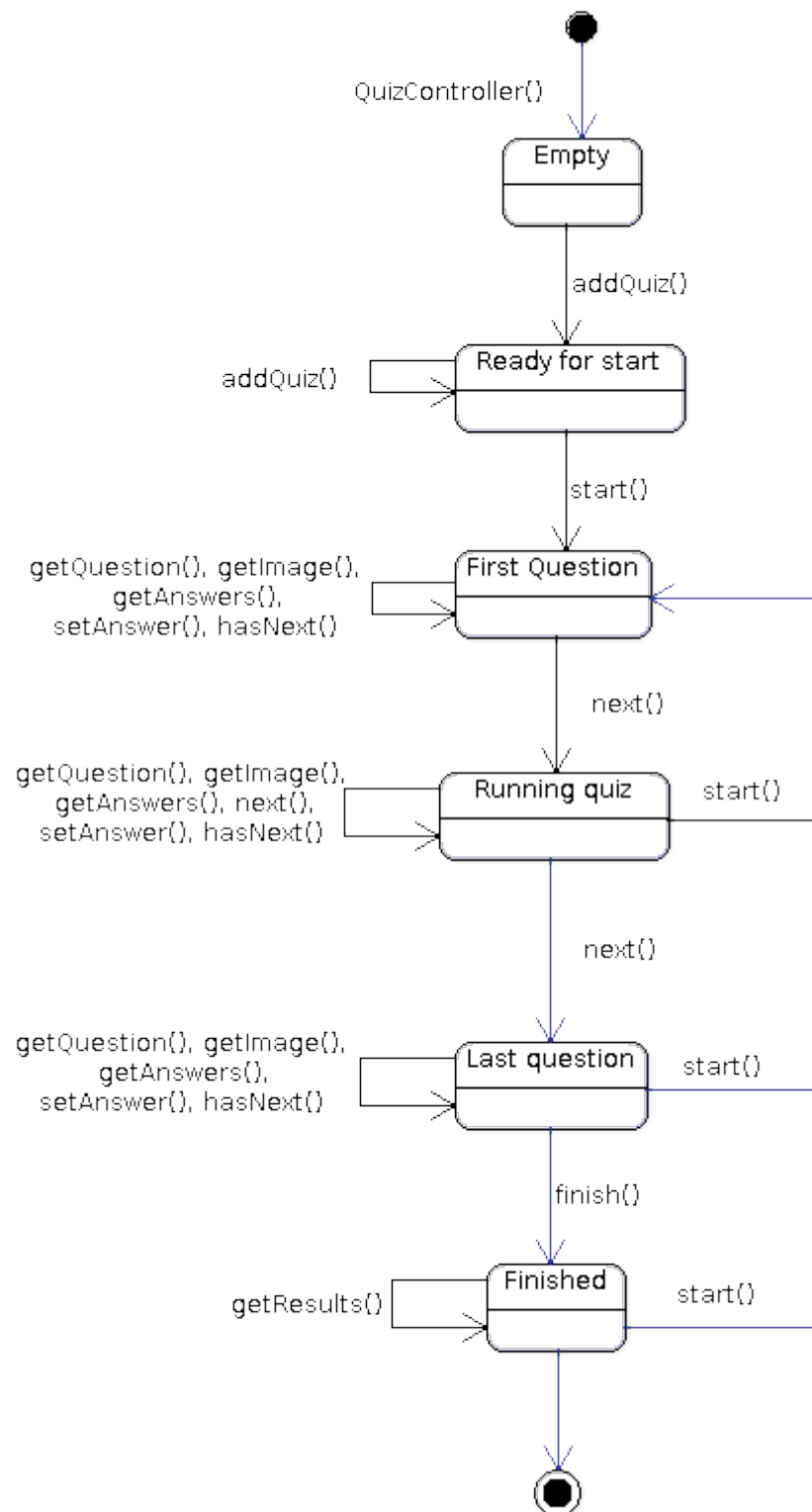
La gestión del quiz la hará un *QuizController* desde la propia Actividad. El *QuizController* trabaja sobre un *QuizDataItem* que almacena los datos de un quiz. Estos datos se añaden a *QuizActivity* mediante el método *setQuiz()*.

Una vez *QuizActivity* disponga del *QuizDataItem*, se creará un *QuizController* con los datos y es el que se usará a lo largo del quiz.

```
public void setQuiz(QuizLevelDataItem quiz){
    QuizController qController = new QuizController();
    qController.addQuiz(quiz);
    this.quiz = qController;
}
```

Guía de QuizController

QuizController.java es una clase que ayuda a gestionar y utilizar un quiz a partir de los datos de un QuizLevelDataItem. QuizController dispone de varios métodos que deben ser utilizados teniendo en cuenta el flujo esperado del quiz. A continuación se muestran los posibles estados de QuizController y qué métodos son seguros en cada uno de ellos.



QuizActivity es una ventana donde se muestra una descripción del quiz y un botón para comenzar. El botón crea una nueva ventana donde se mostrarán las preguntas y las respuestas.

```
Button start = (Button) findViewById(R.id.startQuizButton);

start.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent (getApplicationContext(), QuizQuestionsActivity.class);
        startActivity(i);
    }
});
```

La variable de *QuizController* que se ha creado previamente es un atributo static de *QuizActivity* y es accesible desde el resto de ventanas (de resultados y de preguntas).

```
static QuizController quiz;
```

Desde los *onCreate()* de *QuizQuestionActivity* y de *QuizResultsActivity* se autoconfigura la ventana para trabajar con la variable *quizController*.

QuizQuestionActivity utiliza métodos para actualizar la pregunta en la ventana hasta que el quiz termine. Cuando esto ocurra, aparecerá la ventana de resultados que mostrará los resultados a partir del mismo *QuizController*.

Por último, la pantalla *QuizResultsActivity* muestra los resultados en una barra de progreso. Esta barra de progreso es configurable según cómo se quieran utilizar los resultados que devuelve *QuizController*. Para mostrar los resultados se utiliza el método *updateScoreBar*. A continuación se muestra un ejemplo donde se multiplica por 1000 los resultados devueltos por *QuizController*.

```
private void updateScoreBar(){
    ProgressBar score = (ProgressBar) findViewById(R.id.score);
    TextView scoreText = (TextView) findViewById(R.id.scoreText);
    score.setMax(results.getTotalWeight()*1000);
    if(score.getProgress()<results.getScore()*1000){
        score.incrementProgressBy((int) (results.getScore()*1000/10));
        scoreText.setText(String.valueOf(score.getProgress())
+ "/" + results.getTotalWeight()*1000);
    }else{
        scoreText.setText(String.valueOf((int)results.getScore()*1000)+ "/" + results.getTotalWeight()*1000);
    }
    handler.postDelayed(runnable, 100);
}
```


Ventana de Vídeo

Descripción

La ventana de vídeo permite reproducir vídeo desde una url. El vídeo ocupa la pantalla completa (sin incluir los menús de navegación y banners de publicidad).

Clases y archivos utilizados

Nombre de la clase	Descripción
VideoLevelDataItem	Almacena todos los datos de una ventana de vídeo
VideoListActivity	Actividad principal de vídeo.
VideoActivityGenerator	Generador de la actividad de Vídeo
video_only.xml	Layout de la ventana de vídeo.

Permisos necesarios

Permiso	Descripción
<code>android.permission.INTERNET</code>	Necesario para descargar vídeo.

Funcionamiento / Generator

La reproducción de vídeo se lleva a cabo mediante un view de vídeo proporcionado por Android. Más información de `VideoView`: <http://developer.android.com/reference/android/widget/VideoView.html>

La ventana de vídeo se configura desde `VideoActivityGenerator`, en el método `loadAppLevelData`.

```
if(hasLength(item.getVideoPath())){
    Log.d("URL",item.getVideoPath());
    VideoView video = (VideoView)activity.findViewById(R.id.video);
    video.setVideoURI(Uri.parse(item.getVideoPath()));
    MediaController mc = new MediaController(activity);
    video.setMediaController(mc);
    video.requestFocus();
    video.start();
    mc.show();
}
```

Se crea un `MediaController` para manejar el `VideoView` y también se inicializa con el método `start()`.

Ventana Web

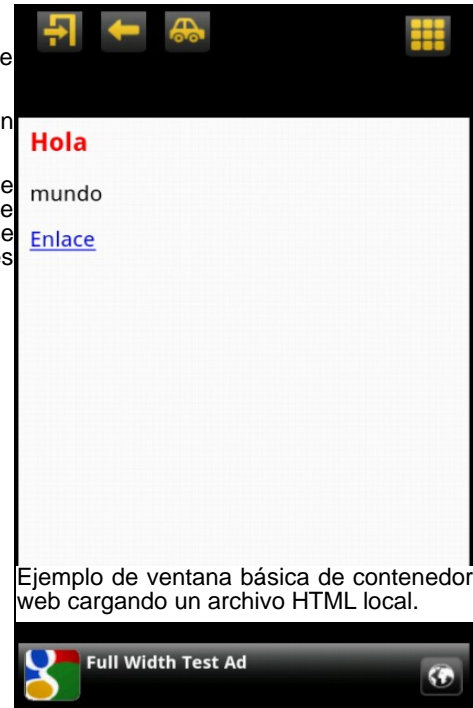
Descripción

La ventana se compone de un WebView donde cargar una url.

La ventana permite no solo cargar una dirección web externa sino que puede cargar también archivo html guardados localmente en el proyecto.

Además, si quiere realizar una aplicación completamente remota con contenidos web puede utilizar esta ventana.

La carga de archivos html locales está pensada para mostrar fácilmente texto con formato. Usted puede cargar cualquier html pero no se recomienda generar aplicaciones que basen su IU en archivos html ya que puede haber problemas en cómo se muestran los contenidos en diferentes tipos de pantallas.



Ejemplo de ventana básica de contenedor web cargando un archivo HTML local.

Clases y archivos utilizados

Nombre de la clase	Descripción
WebLevelDataItem	Almacena todos los datos de una lista
ImageListActivity	Actividad principal de la ventana web. Se reutiliza ImageListActivity
WebActivityGenerator	Generador de la actividad de la ventana web
pdf_activity.xml	Layout de un elemento de la ventana web

Permisos necesarios

Permiso	Descripción
<i>android.permission.INTERNET</i>	Necesario para descargar las imágenes de internet.

Funcionamiento / Generator

Toda la ventana se gestiona desde WebActivityGenerator. El funcionamiento se divide en dos posibles situaciones. Si se va a cargar un archivo html local o si se cargará una dirección web en internet.

Si el archivo que se va a cargar es un html local, el xml de información y por lo tanto el WebLevelDataItem tendrá como valor en dirección url el nombre del archivo (por ejemplo: "archivo.html"). Este archivo tiene que estar disponible en la carpeta "assets/html" del proyecto Android.

Para cargar el archivo local se utiliza el string "file:///android_asset/html/" seguido por el nombre del archivo para que el WebView cargue el html contenido en la carpeta html dentro de assets.

```
if (item.isLocal()){  
    String fileName = item.getWebUrl();  
    WebView webview =  
(WebView)activity.findViewById(R.id.web_viewer);  
    webview.loadUrl("file:///android_asset/html/" + fileName);  
}
```

En el caso en el que se quiera cargar una dirección web, simplemente se utilizará el WebView para que la cargue.

```
else{  
    URL url = new URL(item.getWebUrl());  
    URI uri = new URI(url.getProtocol(), url.getUserInfo(),  
url.getQuery(), url.getHost(), url.getPort(), url.getPath(),  
url.getRef());  
    url = uri.toURL();  
  
    WebView webView =  
(WebView)activity.findViewById(R.id.web_viewer);  
    webView.loadUrl(url.toString());  
}
```

Guía de programación

Cómo crear un nuevo tipo de ventana para el proyecto Android

El siguiente documento es una guía de programación para crear un nuevo tipo de ventana. Muestra en detalle los archivos y métodos que hay que modificar para conseguir que el framework trabaje con el nuevo tipo de ventana apoyándose en un ejemplo concreto (ventana de calendario).

Puede consultar también la “guía rápida nuevo tipo de ventana”.

Paso 1: Tipo de actividad

La nueva ventana tiene su propio tipo de Activity. Existe un enumerado en el paquete *com.emobc.android*, en la clase *ActivityType.java*. Este enumerado se utiliza para comprobar, más adelante, durante la ejecución y creación de pantallas, de qué tipo es la actividad que queremos crear.

Para añadir un nuevo tipo de ventana es necesario, por tanto, modificar el enumerado para incluir el nuevo valor correspondiente a nuestra nueva Actividad.

La nomenclatura utilizada es *ACTIVITY_NAME_ACTIVITY*.

Donde se sustituye *ACTIVITY_NAME* por el nombre de la actividad.

Durante esta guía vamos a crear un nuevo tipo de ventana de calendario. Para este ejemplo crearemos un nuevo caso dentro de *ActivityType.java* (ver figura 1).

```
public enum ActivityType {  
    COVER_ACTIVITY,  
    IMAGE_TEXT_DESCRIPTION_ACTIVITY,  
    IMAGE_LIST_ACTIVITY,  
    LIST_ACTIVITY,  
    VIDEO_ACTIVITY,  
    IMAGE_ZOOM_ACTIVITY,  
    IMAGE_GALERY_ACTIVITY,  
    WEB_ACTIVITY,  
    QR_ACTIVITY,  
    FORM_ACTIVITY,  
    MAP_ACTIVITY,  
    PDF_ACTIVITY,  
    SOUND_ACTIVITY,  
    CALENDAR_ACTIVITY,  
    QUIZ_ACTIVITY,  
    AUDIO_ACTIVITY  
}
```

Paso 2: Crear y definir un xml

El nuevo tipo de ventana tiene que estar definido previamente y esto incluye que el panel de control también tiene que saber que existe. Por lo tanto, desde el panel de control se podrá crear esta nueva ventana y se creará un XML con los datos e información necesarios para generarla desde el proyecto Android.

Figura 1. ActivityType.java

Este archivo XML va a contener los datos para poder crear la ventana. Sin embargo, no tiene porqué ser igual a otros archivos XML. Por ejemplo, una ventana de galería no necesita los mismos datos que una ventana de vídeo. Nuestra nueva ventana puede necesitar una estructura y datos concretos que la diferencian de las demás. Es necesario definir una nueva estructura para esta ventana.

En el caso del Calendario se ha decidido incluir los siguientes datos (ver figura 2):

- *Datald*: para identificar los datos
- *headerImageFile*: Imagen de la cabecera
- *headerText*: Texto de la cabecera

- events: Lista de eventos del calendario
 - event: Evento concreto.
 - title: Nombre del evento
 - eventDate: Fecha del evento
 - text: Descripción del evento
 - NextLevel: Acción al hacer click

```
<levelData>
  <data>
    <dataId>calendar</dataId>
    <headerImageFile>images/white_title.png</headerImageFile>
    <headerText>Información</headerText>
    <events>
      <event>
        <title>Matriculación universidad</title>
        <eventDate>24/07/2012</eventDate>
        <time>23:00</time>
        <text>Consulta el pdf</text>
        <Next Level>
          <nextLevelLevelId>pdf</nextLevelLevelId>
          <nextLevelDataId>pdf</nextLevelDataId>
        </Next Level>
      </event>
    </events>
  </data>
</levelData>
```

Figura 2. Ejemplo de un xml de datos para la ventana de calendario

Paso 3: Crear datos de la aplicación

Una vez tenemos creada y definida la estructura que tendrán los XML que el panel de control creará cada vez que tengamos una ventana de nuestro tipo, es necesario implementar las clases que definan el tipo de datos en tiempo de ejecución. Estas clases se crearán una vez parseamos el XML y contendrán la misma información. La diferencia es que estas clases serán utilizadas durante la ejecución de la aplicación para adquirir más fácil y rápidamente los datos sin tener que parsear continuamente los XML.

Se pueden crear todas las clases que sean necesarias para almacenar los datos, pero es necesario que exista un **LevelDataItem* que es el que almacenará todo el conjunto de datos necesario para la ventana. Esta clase tiene que heredar de *AppLevelDataItem* para tener atributos básicos necesarios para el framework.

Para el ejemplo del Calendario se han creado las siguientes clases que modelan el tipo de datos que utilizará (eventos):

- *CalendarLevelDataItem.java*: Es la clase principal que almacena datos de un calendario (lista de Eventos). Extiende de *AppLevelDataItem* para tener algunos atributos generales como *headerText*.
- *EventDataItem.java*: Clase para almacenar los datos relativos a un evento como son título, descripción o fecha.

Todas estas clases se almacenan en *neurowork.mobile.android.fw.levels.impl*.

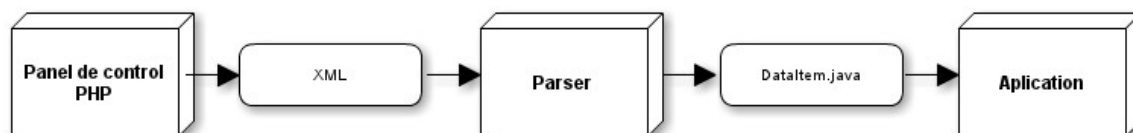
```
public class CalendarLevelDataItem extends AppLevelDataItem {
    //key: date in string dd/mm/yy
    private HashMap<String,TreeSet<EventDataItem>> events;
    public CalendarLevelDataItem(){
        this.events =new HashMap<String,TreeSet<EventDataItem>>();
    }
    public HashMap<String,TreeSet<EventDataItem>> getEvents() {
        return events;
    }
    public void setEvents(HashMap<String,TreeSet<EventDataItem>> events) {
        this.events = events;
    }
}
```

Figura 3. Clase que almacena los datos para la ventana del calendario.

Conviene recordar que todas estas clases deberían tener métodos *get* y *set*, sobre todo los *LevelDataItem*. Los métodos *set* se utilizarán desde *ParseUtils* durante el parseo del xml, y los métodos *get* serán necesarios más adelante para recuperar la información desde los generadores de las ventanas de la aplicación.

Paso 4: Añadir parser

Una vez tenemos definido el XML (paso 2) que contendrá los datos, y las clases que almacenarán estos datos durante la ejecución de la aplicación (paso 3), falta definir el elemento que transformará de uno al otro: el parser.



El parser está implementado en *neurowork.mobile.android.fw.parse*, concretamente en *ParseUtils.java*. Esta clase utiliza una serie de constantes de tipo String para identificar los campos en los archivos XML.

Si nuestra nueva estructura de XML para el nuevo tipo de ventana utilizara campos nuevos que ninguna otra ventana utiliza, sería necesario definir las nuevas constantes correspondientes a estos campos en la clase *ParseUtils.java*.

En el calendario se han definido las siguientes constantes relativas a los datos guardados en la estructura del XML (paso 2):

- a. `private static final String _EVENTS = "events";`
- b. `private static final String _EVENT = "event";`
 - `private static final String _EVENT_DATE = "eventDate";`

Además, el parser del calendario utiliza algunas constantes ya existentes que no ha sido necesario añadir:

- `_DATA_ID_TAG_`
- `_HEADER_IMAGE_FILE_TAG_`
- `_HEADER_TEXT_TAG_`
- `_TITLE_FIELD_` (Título del evento)
- `_TEXT_TAG_` (Descripción del evento)
- `_NEXT_LEVEL_TAG_`
- `_LEVEL_NUMBER_TAG_`
- `_NL_LEVEL_ID_TAG_`
- `_DATA_NUMBER_TAG_`
- `_NL_DATA_ID_TAG_`

Una vez añadidas las constantes necesarias para nuestro XML concreto, es necesario definir un nuevo método en *ParserUtils.java* que implementará el parser específico para el nuevo tipo de Actividad.

La nomenclatura utilizada para estos métodos es:

```
private static Map<String, Object> parse<ActivityName>LevelDataFile(XmlPullParser xpp)
```

El Map de retorno es un mapa que contiene los valores recuperados del XML.

La implementación del método contiene los siguientes pasos

- a. La creación de un `NwXmlStandarParser` específico para nuestro XML a partir del xpp y un nuevo `NwXmlStandarParserTextHandler` con los atributos necesarios para almacenar y los datos reconocidos. Siempre será necesario definir un atributo `AppLevelData` que almacenará el dato.

```
private static Map<String, Object> parse****LevelDataFile(XmlPullParser xpp) {
    final Map<String, Object> ret = new HashMap<String, Object>();

    NwXmlStandarParser parser = new NwXmlStandarParser(xpp,
        new NwXmlStandarParserTextHandler() {
            private AppLevelData appLevelData = new
DefaultAppLevelData();

            private ****LevelDataItem currItem;

            @Override
            public void handleText(String currentField, String text) {
                if(currentField.equals(_DATA_TAG_)){
                    currItem = new ****LevelDataItem();
                    appLevelData.addItem(currItem);

                }else if(currentField.equals(_DATA_ID_TAG_)){
                    currItem.setId(text);

                }else
            if(currentField.equals(_HEADER_IMAGE_FILE_TAG_)){
                currItem.setHeaderImageFile(text);
            }else if(currentField.equals(_HEADER_TEXT_TAG_)){
                currItem.setHeaderText(text);
            }else{
                ret.put(currentField, text);
            }
        }

        @Override
        public void handleEndTag(String currentField) {
            if(currentField.equals(_LEVEL_DATA_TAG_)){
                ret.put(_LEVEL_DATA_TAG_, appLevelData);
                appLevelData.reIndex();
            }
        }

        @Override
        public void handleBeginTag(String currentField) {

        }

    }, _LEVEL_DATA_TAG_);

    parser.startParsing();

    return ret;
}
```

Figura 4. Plantilla básica del método para parsear un archivo xml. En ParseUtils.java

- Sobrescribir algunos métodos de nuestra clase `NwXmlStandarParserTextHandle` y donde implementaremos la verdadera funcionalidad del parser.

Para la creación del nuevo parser es útil partir de uno ya construido y modificarlo para que funcione con los datos concretos del nuevo archivo xml.

La plantilla básica de un método parser puede verse en la figura 4.

Para el calendario ha sido necesario incluir las siguientes variables para almacenar las referencias a los datos que están

siendo leídos del archivo xml y que, una vez terminado, se almacenarán dentro del AppLevelData correspondiente.

```
private CalendarLevelDataItem currItem;

HashMap<String,TreeSet<EventDataItem>> private
currEventsMap;
private TreeSet<EventDataItem> currTree;
private EventDataItem currEvent;
private NextLevel Next Level;
```

También es necesario sobrescribir el método *handleText* donde se incluirá la funcionalidad del parser. En la figura 5 puede verse el la clase *NwXmlStandarParserTextHandle* completa para el calendario. En negrita aparecen los fragmentos añadidos sobre la plantilla de la figura 4.

Ahora que ya se ha creado el método específico que parseará nuestro XML, es necesario especificar cuándo vamos a utilizarlo. Cuando se especifique (en el app.xml) de qué tipo es la actividad, tendremos que llamar al método parser específico. Para concretar esto es necesario añadir un nuevo caso para el nuevo tipo de ventana en el método *parseLevelDataFile*, también en *ParseUtils*.

En el calendario se ha incluido el siguiente caso:

```
case CALENDAR_ACTIVITY:

    return parseCalendarLevelDataFile(xpp);
```

Paso 5: Añadir Activity

En Android cada ventana tiene su propia Activity. A la hora de crear un nuevo tipo de ventana conviene preguntarse si es necesario crear una nueva Activity o podemos reutilizar alguna de las existentes. Por defecto el framework utiliza *ImageListActivity.java* que es una actividad básica sobre la que se podría utilizar un layout concreto.

Si preferimos crear una nueva actividad porque las existentes no cumplen con nuestras necesidades, o necesitamos algunos métodos concretos que solo se utilizarán en nuestra Activity, tendremos que incluirla en el paquete *net.neurowork.mobile.fw.activities*.

La nomenclatura utilizada es: *<ActivityName>Activity.java*

La nueva actividad tiene que heredar de *createMenus*. *CreateMenus* es una clase que hereda de *Activity* e incluye métodos para generar los menús de navegación y los banners de publicidad que todas las ventanas de la aplicación deberían tener.

La plantilla básica para crear una clase Activity nueva es la de *ImageListActivity.java* de la figura 6.

Figura 4. Plantilla básica del método para parsear un archivo xml. En *ParseUtils.java*

Figura 5. Parser para el archivo xml del calendario.

```
public class ImageListActivity extends CreateMenus {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        boolean isEntryPoint = false;
        boolean isSideMenu = false;
        ApplicationData applicationData = SplashActivity.getApplicationData();
        if(applicationData != null){
            Intent intent = getIntent();
            isEntryPoint=(Boolean)intent.getSerializableExtra(ApplicationData.IS_ENTRY_POINT_TAG);
            isSideMenu=(Boolean)intent.getSerializableExtra(ApplicationData.IS_SIDE_MENU_TAG);
            isEntryPoint=(Boolean)intent.getSerializableExtra(ApplicationData.IS_ENTRY_POINT_TAG);
            NextLevel Next Level =
            (NextLevel)intent.getSerializableExtra(ApplicationData.NEXT_LEVEL_TAG);
            ActivityGenerator generator = applicationData.getFromNextLevel(this, Next Level);
            generator.initializeActivity(this);
        }
        if(isSideMenu == false){
            createMenus(isEntryPoint);
        }else{
            createSideBar();
        }
    }
}
```

Figura 6. Plantilla para crear un nuevo Activity

IMPORTANTE: Si se crea una nueva Activity, es necesario declararla en Manifest.xml

Una vez creada o reutilizada la Activity que usaremos para el nuevo tipo de ventana, es necesario añadir un nuevo caso al método *getActivityClass* en *AppLevel.java* dentro del paquete *neurowork.mobile.android.fw.levels* donde se devolverá la clase de la actividad.

Por último, la actividad tendrá un layout concreto. Es necesario definirlo dentro de la carpeta *layouts* en la carpeta *res*.

Paso 6: Crear ActivityGenerator

En el framework las actividades son creadas por los generadores de actividades. Durante la ejecución de la aplicación, cuando se vaya a crear una ventana, el generador específico será llamado para encargarse esa tarea.

Por cada tipo de ventana tiene que existir un generador asociado en el paquete:

net.neurowork.mobile.fw.activities.generators.

La nomenclatura utilizada es: *<ActivityName>Generator.java*

```

public class ****ActivityGenerator extends LevelActivityGenerator {

    public ****ActivityGenerator(AppLevel appLevel, NextLevel Next Level) {
        super(appLevel, Next Level);
    }

    @Override
    protected void loadAppLevelData(final Activity activity, final AppLevelData data) {
        final ****LevelDataItem item = (****LevelDataItem)data.findByNextLevel(Next
Level);

        initializeHeader(activity, item);

        //Create Banner
        CreateMenus c = (CreateMenus)activity;
        c.createBanner();
    }

    @Override
    protected int getContentViewResourceId(final Activity activity) {
        if(appLevel.getXib() != null && appLevel.getXib().length() > 0){
            int id = getActivityLayoutIdFromString(activity, appLevel.getXib());
            if(id > 0)
                return id;
        }
        return R.layout.***;
    }
}

```

Figura 7. Plantilla para crear un nuevo ActivityGenerator

La plantilla básica sobre la que crear un nuevo ActivityGenerator está definida en la figura 7. A continuación se explican en detalle los puntos más importantes de esta plantilla aplicados al ejemplo del calendario.

El método *loadAppLevelData* recupera la información de la ventana desde el ApplicationData (es el objeto que almacena todos los datos de la aplicación, para más información ver documento levels) de la aplicación. La información recuperada se almacena en una variable *item*, del mismo tipo de la clase de datos que se ha creado en el paso 3. El método *loadAppLevelData* es llamado desde el *onCreate()* de cada Activity (ver figura 6) y encapsula toda la lógica necesaria para crear la ventana (layouts, onClickListener...).

Dentro de este método, si estamos creando una ventana de Calendario, habría que modificar el item para que sea del tipo CalendarLevelDataItem y poder trabajar con los métodos y atributos específicos como los métodos get para recuperar la información.

```

final CalendarLevelDataItem item = (CalendarLevelDataItem)data.findByNextLevel(Next Level);

```

El método *initializeHeader* y *createBanner* se encargan de crear una cabecera a la ventana e inicializar los banner de publicidad, si los hubiera, respectivamente.

```
initializeHeader(activity, item);

//Create Banner
CreateMenus c =
(CreateMenus)activity;
c.createBanner();
```

Ahora podemos recuperar todos los elementos de los que dispone el layout de nuestra ventana (diseñado en el paso 5) y configurarlos personalmente.

El layout del calendario dispone de un CalendarView que es la cuadrícula donde aparecen los días del mes, una lista de eventos, y dos botones para cambiar de mes y un TextView donde aparece el mes que estamos viendo.

```
final CalendarView cv = (CalendarView)activity.findViewById(R.id.calendarView);
final TextView tv = (TextView) activity.findViewById(R.id.textViewMonth);
Button prev = (Button) activity.findViewById(R.id.prevMonthButton);
Button next = (Button) activity.findViewById(R.id.nextMonthButton);
ListView lv = (ListView) activity.findViewById(R.id.listViewCalendarEvents);
```

La recuperación de estos View se realiza mediante la variable activity, que es la actividad que ha llamado al generator. Después de la recuperación de los datos es posible configurarlos como cualquier View en Android.

Si, por ejemplo, quisiéramos añadir funcionalidad a un botón, deberíamos incluir el siguiente fragmento de código después de recuperar la variable del botón.

```
final CalendarView cv = (CalendarView)activity.findViewById(R.id.calendarView);
final TextView tv = (TextView) activity.findViewById(R.id.textViewMonth);
Button next = (Button) activity.findViewById(R.id.nextMonthButton);
next.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        cv.nextMonth();
        tv.setText(h.getMonthInString(cv.getMonth()+1));
    }
});
```

Este ejemplo añade la funcionalidad al botón de siguiente mes para actualizar el calendarView y el TextView para que muestren el siguiente mes. Nótese que las variables cv y tv deben ser finales para poder acceder a ellas desde el ClickListener que se ha creado para el botón.

También sería posible incluir una llamada a un método declarado dentro del ActivityGenerator.

```

AppLevelData data = appLevel.getData(activity);
CalendarLevelDataItem item = (CalendarLevelDataItem)data.findByNextLevel(Next Level);
HashMap<String,TreeSet<EventDataItem>> TotalEvents = item.getEvents();
TreeSet<EventDataItem> eventsInDate = TotalEvents.get(date);
ArrayList<EventDataItem> eventsList = new ArrayList<EventDataItem>();

//Maybe null
if (eventsInDate!=null){
    eventsList.addAll(eventsInDate);
}

//Putting the list in the ListView
ListView lv = (ListView)activity.findViewById(R.id.listViewCalendarEvents);
lv.setAdapter(new CallListAdapter(activity, R.layout.list_item, eventsList));
lv.setTextFilterEnabled(true);

```

Figura 8. Cómo utilizar LevelDataItem en la configuración de los View de un layout.

Para el caso de la lista de eventos, la información con la cual configurar el View está disponible dentro de nuestra variable *item*, del tipo *CalendarLevelDataItem*. Si queremos recuperar dicha información del *item* tendremos que usar los métodos *get* definidos en el paso 3.

En la figura 8 se muestra cómo el calendario gestiona la lista de eventos contenida en la variable *item*.

Para terminar con el *ActivityGenerator* es importante también el método *getContentViewResourceId* de la plantilla de la figura 7. Es el encargado de devolver el layout específico para nuestra ventana.

```

@Override
protected int getContentViewResourceId(Activity activity) {
    if(appLevel.getXib() != null && appLevel.getXib().length() > 0){
        int id = getActivityLayoutIdFromString(activity, appLevel.getXib());
        if(id >0)
            return id;
    }
    return R.layout.calendar_layout;
}

```

Figura 9. Método que devuelve el layout. En *ActivityGenerator*.

En la figura 9 se muestra cómo es el método para la ventana de calendario, y cómo está asignado por defecto el layout disponible en la carpeta *res*.

Una vez finalizado el *ActivityGenerator*, debemos incluirlo dentro de la clase *ActivityGeneratorFactory* en el método *createActivityGenerator()*, añadiendo un nuevo caso. Retomando el ejemplo del calendario, habría que incluir el siguiente caso:

```
public static ActivityGenerator createActivityGenerator(Context context, NextLevel Next Level) throws
InvalidFileException{
    ApplicationData applicationData = SplashActivity.getApplicationData();
    AppLevel appLevel = applicationData.getNextAppLevel(Next Level);
    switch (appLevel.getActivityType()) {
        case IMAGE_TEXT_DESCRIPTION_ACTIVITY:
            return new ImageTextDescriptionActivityGenerator(appLevel, Next Level);
        [...]
        case PDF_ACTIVITY:
            return new PdfActivityGenerator(appLevel, Next Level);
        case CALENDAR_ACTIVITY:
            return new CalendarActivityGenerator(appLevel, Next Level);
        case QUIZ_ACTIVITY:
            return new QuizActivityGenerator(appLevel, Next Level);
        case AUDIO_ACTIVITY:
            return new AudioActivityGenerator(appLevel, Next Level);
        default:
            break;
    }
    Log.e("eMobc", "No existe el activity generator para el tipo: " +
appLevel.getActivityType().toString());
    return null;
}
}
```

Anexo: Actividad de terceros

A continuación se detallan los pasos básicos para utilizar una actividad de terceros como nuevo tipo de ventana.

1. Realizar los pasos 1, 2, 3, y 4 de la guía de programación para un nuevo tipo de ventana.
2. Añadir la actividad correspondiente a la nueva ventana en el paquete *net.neurowork.mobile.activities*
3. Añadir todas las clases necesarias dentro de un paquete nuevo u otro paquete del proyecto.
4. Añadir los layouts necesarios en la carpeta *res/layout*.
5. Crear un ActivityGenerator para la nueva pantalla como se detalla en la guía de programación en el paso 6, modificando el método *getContentViewResourceId* para que devuelva por defecto el layout del paso 3.
Si se requieren datos de xml, utilizar el método *loadAppLevelData* para configurar los Views del layout.
6. Añadir la lógica de la plantilla de la figura 6 en el método *onCreate()* de la Actividad de terceros.
7. Si es posible y se considera necesario, sustituir la lógica previa del *onCreate()*, así como del resto de la Activity al nuevo ActivityGenerator, teniendo en cuenta las siguientes referencias:

Acciones	Activity	Generator (loadAppLevelData)
Acceder a un recurso (View)	<code>findViewById(R.id.view);</code>	<code>(View) activity.findViewById(R.id.view);</code>
Cargar un layout en la Activity	<code>setContentView(R.layout.main);</code>	Sobrescribir le método <code>getContentViewResourceId()</code> para que devuelva por defecto <code>R.layout.main</code> (ver figura 9, paso 6)

Crear un nuevo tipo de ventana (Android)

Si necesita crear un nuevo tipo de ventana porque ninguna de las existentes se adapta a sus necesidades, puede crear uno nuevo siguiendo los pasos que se muestran a continuación.

El siguiente documento es una guía rápida para preparar el proyecto de Android al nuevo tipo de pantalla. Si quiere crear un nuevo tipo para otros proyectos debería consultar los documentos correspondientes. También conviene consultar el documento para crear un nuevo tipo de ventana en el panel de control.

Para más información de cómo crear un nuevo tipo de pantalla, así como ver un ejemplo concreto, puede ver la “guía para crear un nuevo tipo de ventana”.

Paso 1: Tipo de actividad

Añadir nuevo caso al enumerado `ActivityType` dentro del paquete `com.emobic.android`. El nuevo caso es el nuevo tipo de pantalla.

`ACTIVITY_NAME_ACTIVITY`

Paso 2: Crear y definir un xml

Es necesario definir el xml para el nuevo tipo de ventana. Tendría que incluir todos los datos necesarios para poder construir la ventana. Es posible reutilizar alguno existente.

Paso 3: Crear datos de la aplicación

Crear los datos que se usarán en tiempo de ejecución por la aplicación. Estos datos serán clases que almacenan toda la información contenida en los xml previamente definidos.

Las clases de datos están almacenadas en *neurowork.mobile.android.fw.levels.impl*.

`<NAME>LevelDataItem.java`

`<NAME>DataItem.java`

Paso 4: Añadir parser

- Crear un nuevo método dentro de *ParseUtils.java* para parsear el archivo xml del nuevo tipo de ventana.
`private static Map<String, Object> parse<ACTIVITY_NAME>LevelDataFile(XmlPullParser xpp)`
- Añadir el nuevo caso dentro del método *parseLevelDataFile* en *ParseUtils.java* para que devuelva el nuevo parser creado ante el nuevo tipo de ventana.

Paso 5: Añadir Activity

- Si fuera necesario, crear una nueva activity para el nuevo tipo de ventana que herede de createMenus.

`<ActivityName>Activity.java`

Es posible reutilizar una activity previamente creada si se adapta a las necesidades del nuevo tipo de ventana.

IMPORTANTE: Si se crea una nueva Activity, es necesario declararla en Manifest.xml

- Añadir el nuevo caso al método `getActivityClass` en `AppLevel.java` dentro del paquete `neurowork.mobile.android.fw.levels` donde se devolverá la clase de la actividad creada o la que se reutilizará.
- Crear el layout correspondiente al nuevo tipo de ventana.

`<activity_type>_layout.xml`

Paso 6: Crear ActivityGenerator

- Crear el nuevo generador de ventanas para el nuevo tipo de ventana dentro del paquete `net.neurowork.mobile.fw.activities.generators`

`<ActivityName>Generator.java`

El generator utilizará el tipo de datos del paso 3 para configurar el layout del paso 5.

- Incluir este nuevo generador en la clase `ActivityGeneratorFactory`, en el método `createActivityGenerator()`, añadiendo un nuevo caso.

Cómo implementar un NextLevel (Android)

La implementación del enlace hacia un next level está contenida en el método `showNextLevel` de `AbstractActivityGenerator`. Los parámetros de este método son:

- `Activity`: Contexto desde donde se lanzará la nueva ventana
- `NextLevel`: `NextLevel` asociado que se quiere lanzar.

El método `showNextLevel` se encargará de buscar el level en el `AplicationData` para cargar los datos y generar la nueva ventana.

```
protected void loadAppLevelData(final Activity activity, final AppLevelData data) {  
    LinearLayout leerMasBtn =(LinearLayout) activity.findViewById(R.id.descr_mas);  
    leerMasBtn.setOnClickListener(new View.OnClickListener() {  
        public void onClick(View view) {  
            showNextLevel(activity, item.getNextLevel());  
        }  
    });  
}
```

Ejemplo de cómo implementar la transición a un Next Level en la pulsación de un botón desde un Generator en el método `loadAppLevelData`.